Univerzitet Singidunum

Miodrag Živković

RAZVOJ MOBILNIH APLIKACIJA

ANDROID JAVA PROGRAMIRANJE

Beograd, 2020.

UNIVERZITET SINGIDUNUM

Miodrag Živković

RAZVOJ MOBILNIH APLIKACIJA

Android Java programiranje

Prvo izdanje

Beograd, 2020.

RAZVOJ MOBILNIH APLIKACIJA

Android Java programiranje

Autor:
dr Miodrag Živković

Recenzenti:
Prof. Aleksandar Jevremović, redovni profesor, Univerzitet Singidunum
Prof. Nebojša Bačanin Džakula, vanredni profesor, Univerzitet Singidunum
Prof. Boško Nikolić, redovni profesor, Elektrotehnički fakultet Univerziteta u Beograd u

Izdavač:
UNIVERZITET SINGIDUNUM
Beograd, Danijelova 32
www.singidunum.ac.rs

Za izdavača:
dr Milovan Stanišić

Priprema za štampu:
Miodrag Živković

Dizajn korica:
Aleksandar Mihajlović

Godina izdanja:
2020.

Tiraž:
700 primeraka

Štampa:
Caligraph, Beograd

ISBN: 978-86-7912-719-8

Copyright:
© 2020. Univerzitet
Singidunum Izdavač zadržava
sva prava.
Reprodukcija pojedinih delova ili celine ove publikacije nije dozvoljena.

Sadržaj

Lista slika.....VII
Predgovor.....XIII
1. Uvod.....1
1.1. Istorijat.....2
1.1.1. Prve verzije.....3
1.1.2. Android 1.5 Cupcake.....4
1.1.3. Android 1.6 Donut.....5
1.1.4. Android 2.0 Eclair.....6
1.1.5. Android 2.2 Froyo.....7
1.1.6. Android 2.3 Gingerbread.....8
1.1.7. Android 3.0 Honeycomb.....10
1.1.8. Android 4.0 Ice Cream Sandwich.....12
1.1.9. Android 4.1 Jelly Bean.....13
1.1.10. Android 4.4 KitKat.....14
1.1.11. Android 5.0 Lollipop.....16
1.1.12. Android 6.0 Marshmallow.....17
1.1.13. Android 7.0 Nougat.....19
1.1.14. Android 8.0 Oreo.....20
1.1.15. Android 9.0 Pie.....21
1.1.16. Android 10.....23
1.2. Rasprostranjenost Android verzija na tržištu.....25
1.3. Udeo Android operativnog sistema na tržištu mobilnih uređaja.....27
1.4. Tržište mobilnih aplikacija.....29
2. Android platforma.....34
2.1. Arhitektura Android platforme.....34
2.2. Android Runtime.....36

2.3. Android aplikacije....39
2.4. Okruženje za razvoj....41
2.5. Android uređaji i emulatori....43
3. Android Studio....47
3.1. Instalacija okruženja....47
3.2. Kreiranje i struktura projekta....55
3.3. Konfigurisanje Emulatora i pokretanje aplikacije....66
3.4. SDK Manager....72
4. Grafički korisnički interfejs....76
4.1. Klasa View....77
4.2. Standardne komponente grafičkog interfejsa....79
4.2.1. TextView....82
4.2.2. EditText....84
4.2.3. Button....87
4.2.4. CheckBox....90
4.2.5. RadioButton....95
4.2.6. Spinner....97
4.2.7. Toast....101
4.2.8. DatePicker i TimePicker....104
4.2.9. Dialog....106
4.3. Rasporedi komponenti (layout)....110
4.3.1. Linearni raspored....112
4.3.2. Relativni raspored....116
4.3.3. Ograničeni raspored....121
4.3.4. ListView....122
4.4. Layout editor....124
4.5. Klasa R....126
4.6. Rukovanje događajima....128

4.7. Pitanja za vežbu....131
5. Aktivnosti....133
5.1. Konfigurisanje manifesta....134
5.2. Životni ciklus aplikacije....135
5.3. Životni ciklus aktivnosti....138
5.4. Čuvanje stanja korisničkog interfejsa....146
5.5. Promena orijentacije ekrana....149
5.6. Intenti (namere)....153
5.6.1. Intent objekat....154
5.6.2. Eksplicitne i implicitne namere....155
5.6.3. Intent filteri i rezolucija Intenta....160
5.7. Back Stack....161
5.8. Pitanja za vežbu....165
5.9. Zadaci za vežbu....166
6. Niti....181
6.1. AsyncTask....183
6.2. Pitanja za vežbu....186
6.3. Zadaci za vežbu....187
7. Dozvole....199
7.1. Tipovi dozvola....199
7.2. Deklaracija privilegija u manifestu....201
7.3. Potvrda opasnih dozvola....202
7.3.1. Potvrda za vreme izvršavanja....202
7.3.2. Potvrda za vreme instalacije....203
7.4. Provera dozvola....204
7.5. Zahtevanje dozvola....204
7.6. Pitanja za vežbu....208
7.7. Zadaci za vežbu....209

8. Fragmenti....222
8.1. Životni ciklus fragmenta....223
8.2. Iscrtavanje korisničkog interfejsa fragmenta....226
8.3. Dodavanje fragmenta aktivnosti....226
8.3.1. Deklaracija fragmenta unutar fajla sa rasporedom komponenti aktivnosti....227
8.3.1. Programsko dodavanje fragmenta....228
8.4. Transakcija fragmenta....230
8.5. Komunikacija fragmenta sa aktivnosti domaćinom....231
8.6. Pitanja za vežbu....236
8.7. Zadaci za vežbu....237
9. Perzistencija podataka....243
9.1. SharedPreferences....243
9.2. Interni prostor....245
9.3. Eksterni prostor....246
9.4. SQLite baza podataka....248
9.4.1. Definisanje šeme i ugovora....248
9.4.2. Kreiranje baze pomoću SQL pomoćne klase....249
9.4.3. Ubacivanje podataka u bazu....251
9.4.4. Čitanje podataka iz baze....252
9.4.5. Brisanje podataka iz baze....253
9.4.6. Ažuriranje podataka u bazi....254
9.4.7. Perzistencija konekcije....255
9.5. Room biblioteka....256
9.6. Pitanja za vežbu....260
9.7. Zadaci za vežbu....261
10. BroadcastReceiver komponente....281
10.1. Sistemska obaveštenja....281

10.2. Prijem obaveštenja....282
10.2.1. Deklaracija prijemnika kroz manifest....282
10.2.2. Programska registracija prijemnika....284
10.3. Slanje obaveštenja....286
10.4. Sigurnosna razmatranja i praktične primene....287
10.5. Pitanja za vežbu....289
10.6. Zadaci za vežbu....290
11. Servisi....308
11.1. Životni ciklus servisa....308
11.2. Deklaracija servisa u manifestu....311
11.3. Startovani servis....312
11.3.1. IntentService klasa....313
11.3.2. Service klasa....314
11.3.3. Pokretanje i zaustavljanje servisa....318
11.4. Vezani servis....319
11.5. Pitanja za vežbu....326
11.6. Zadaci za vežbu....327
12. Praktični primeri upotrebe....350
12.1. Kamera....350
12.2. Lokacija....353
12.3. Senzori....356
12.4. Rad sa Google mapama....360
Literatura....369

Lista slika

Slika 1.1, prvi komercijalno dostupni Android telefon HTC Dream....2
Slika 1.2, Android 1.5 Cupcake glavni ekran....4
Slika 1.3, Android 1.6 Donut glavni ekran....5
Slika 1.4, Android 2.0 Eclair poboljšana navigacija i funkcija speech−to−text....7
Slika 1.5, Android 2.2 Froyo glavni ekran....8
Slika 1.6, Android 2.3 Gingerbread glavni ekran....9
Slika 1.7, Android 3.0 Honeycomb glavni ekran....11
Slika 1.8, Android 3.0 Honeycomb Recent apps....11
Slika 1.9, Android 4.0 Ice Cream Sandwich glavni ekran i Recent Apps....12
Slika 1.10, Android 4.1 Jelly Bean korisnički interfejs....14
Slika 1.11, Android 4.4 KitKat korisnički interfejs....15
Slika 1.12, Android 5.0 Lollipop korisnički interfejs i materijalni dizajn....16
Slika 1.13, Android 6.0 Marshmallow korisnički interfejs....18
Slika 1.14, Android 7.0 Nougat korisnički interfejs i nova opcija podeljenog ekrana....19
Slika 1.15, Android 8.0 Oreo korisnički interfejs i nova opcija slike u slici....21
Slika 1.16, Android 9.0 Pie nova opcija pozicioniranja u zatvorenom prostoru 22
Slika 1.17, Android 10 kontrola aplikacija koje pristupaju lokaciji....23
Slika 1.18, Android 10 podrška za savitljive ekrane....24
Slika 1.19, raspodela različitih verzija Androida na tržištu....25
Slika 1.20, raspodela veličine ekrana Android uređaja na tržištu....27
Slika 1.21, iOS, Android, BlackBerry i Symbian operativni sistemi....28
Slika 1.22, ukupan broj preuzimanja aplikacija na svim platformama....30
Slika 1.23, ukupan broj besplatnih i plaćenih preuzimanja u intervalu 2011−2017....31
Slika 1.24, uporedni broj preuzimanja sa Google Play i App Store....32
Slika 1.25, broj aplikacija dostupnih na Google Play Store....33
Slika 2.1, Android softverski stek....34
Slika 2.2, uporedni prikaz JVM i Dalvik VM....37
Slika 2.3, JIT arhitektura....38
Slika 2.4, izvršavanje Android aplikacija u zasebnim Linux procesima....40
Slika 2.5, struktura APK....41
Slika 2.6, Android Studio IDE....42
Slika 2.7, proces kreiranja aplikacije u Android Studio IDE....43
Slika 2.8, skrivena sekcija Developer options....44

Slika 2.9, Android Emulator sa dodatnim funkcionalnostima.....45
Slika 3.1, verzija Android Studio okruženja korišćena u ovoj knjizi.....47
Slika 3.2, početni ekran procesa instalacije.....48
Slika 3.3, odabir komponenti za instalaciju.....48
Slika 3.4, specifikacija lokacije gde će Android Studio biti instaliran.....49
Slika 3.5, odabir Start Menu direktorijuma.....49
Slika 3.6, proces instalacije Android Studio okruženja.....50
Slika 3.7, završetak instalacije Android Studio okruženja.....50
Slika 3.8, import podešavanja od prethodne verzije (ukoliko postoji).....51
Slika 3.9, Android Studio splash screen.....51
Slika 3.10, preuzimanje potrebnih SDK komponenti prilikom prvog pokretanja okruženja.....51
Slika 3.11, Android Studio čarobnjak za podešavanje okruženja.....52
Slika 3.12, odabir tipa instalacije okruženja.....52
Slika 3.13, odabir teme korisničkog interfejsa okruženja.....53
Slika 3.14, verifikacija odabranih opcija.....53
Slika 3.15, čarobnjak preuzima i instalira SDK komponente, sa i bez prikaza detalja.....54
Slika 3.16, Welcome to Android Studio ekran.....54
Slika 3.17, odabir platforme i šablona aktivnosti za novi projekat.....55
Slika 3.18, osnovno podešavanje projekta.....56
Slika 3.19, odabir druge vrednosti minimalnog podržanog API nivoa.....57
Slika 3.20, procenat uređaja koji će moći da pokrenu aplikaciju na osnovu zadatog minimalnog API nivoa.....57
Slika 3.21, kreirani HelloWorld projekat.....57
Slika 3.22, stuktura standardne Android aplikacije.....58
Slika 3.23, projektni fajlovi u Android pogledu.....58
Slika 3.24, struktura glavnog prozora Android Studio okruženja.....59
Slika 3.25, tab sa XML rasporedom komponenti u editoru.....61
Slika 3.26, lokacija XML fajlova sa rasporedom u strukturi projekta (res/layout) 62
Slika 3.27, grafički editor za XML fajlove sa rasporedom komponenti.....62
Slika 3.28, pregled sadržaja XML koda rasporeda komponenti.....63
Slika 3.29, lokacija AndroidManifest.xml fajla u strukturi projekta.....64
Slika 3.30, build.gradle fajl za modul app i njegova lokacija u strukturi projekta 65
Slika 3.31, sadržaj build.gradle fajla za modul app.....66

Slika 3.32, sekcija Tools, gde se nalaze AVD i SDK Manager opcije.....66
Slika 3.33, SDK Tools tab u okviru SDK Manager sekcije.....67
Slika 3.34, AVD Manager u okviru sekcije Tools.....68
Slika 3.35, Raspoloživi virtuelni uređaji u okviru opcije AVD Manager.....68
Slika 3.36, odabir hardverskog profila novog virtuelnog uređaja.....69
Slika 3.37, odabir sistemske slike virtuelnog uređaja.....69
Slika 3.38, verifikacija odabrane konfiguracije.....70
Slika 3.39, napredna podešavanja virtuelnog uređaja.....70
Slika 3.40, novi uređaj prikazan u AVD Manager ekranu.....71
Slika 3.41, pokretanje aplikacije na dodatom virtuelnom uređaju.....71
Slika 3.42, pokrenuta aplikacija na virtuelnom uređaju.....72
Slika 3.43, lista svih dostupnih SDK platformi.....73
Slika 3.44, lista dostupnih SDK platformi sa detaljima za svaku pojedinačnu platformu.....73
Slika 3.45, lista svih dostupnih SDK alata.....74
Slika 3.46, proces ažuriranja komponenti.....75
Slika 4.1, primer strukture korisničkog interfejsa sa ViewGroup komponentama i pripadajućim View objektima.....76
Slika 4.2, definicija pozicije View komponente.....78
Slika 4.3, padding i margine komponente.....79
Slika 4.4, prikaz tipične upotrebe TextView komponente.....83
Slika 4.5, prikaz tipične upotrebe EditText komponente.....86
Slika 4.6, različiti tipovi virtuelne tastature za različite vrednosti inputType atributa.....87
Slika 4.7, tipičan primer upotrebe dugmeta sa jednostavnom obradom događaja.....90
Slika 4.8, komponente CheckBox.....91
Slika 4.9, tipičan primer upotrebe CheckBox komponenti.....95
Slika 4.10, tipičan primer upotrebe RadioButton komponente.....97
Slika 4.11, tipičan primer upotrebe Spinner komponente.....100
Slika 4.12, prikaz kratke Toast poruke u mail aplikaciji.....101
Slika 4.13, primer upotrebe Toast poruke.....103
Slika 4.14, TimePicker i DatePicker komponente.....104
Slika 4.15, Izgled DatePicker komponente u okviru aplikacije.....105
Slika 4.16, tipični primeri Dialog komponenti.....107
Slika 4.17, prikaz jednostavnog Dialog−a.....109
Slika 4.18, hijerarhija komponenti sa pridruženim layout parametrima.....110

Slika 4.19, najčešće korišćeni rasporedi komponenti.....111
Slika 4.20, najčešće korišćeni rasporedi komponenti koji koriste adapter.....112
Slika 4.21, linearni raspored komponenti.....113
Slika 4.22, primer vertikalnog linearnog rasporeda sa težinama.....115
Slika 4.23, relativni raspored komponenti.....116
Slika 4.24, primer relativnog rasporeda.....120
Slika 4.25, dodavanje ograničenja komponenti u ograničenom rasporedu.…122
Slika 4.26, tipičan primer upotrebe ListView rasporeda komponenti.....124
Slika 4.27, Layout Editor − grafički editor za dizajn rasporeda komponenti....125
Slika 4.28, podešavanje atributa komponente u grafičkom editoru.....126
Slika 4.29, R.java klasa, lokacija i njen sadržaj.....127
Slika 5.1, hijerarhija procesa po važnosti.....136
Slika 5.2, životni ciklus aktivnosti sa callback metodama.....139
Slika 5.3, dijagram stanja u kojima se aktivnost može naći.....142
Slika 5.4, primer aktivnosti koja gubi fokus usled dolaznog poziva.....144
Slika 5.5, uticaj promene rotacije uređaja na stanje aktivnosti.....145
Slika 5.6, dodavanje alternativnog prikaza za Landscape režim.....149
Slika 5.7, dodatni XML fajl za raspored komponenti u Landscape režimu.....150
Slika 5.8, korisnički interfejs aplikacije sa odvojenim rasporedima za Portrait i Landscape režim.....152
Slika 5.9, implicitna namera i poziv aktivnosti kamere.....158
Slika 5.10, prosleđivanje implicitne namere.....158
Slika 5.11, dijalog za odabir aplikacije.....159
Slika 5.12, princip rada back stack−a.....162
Slika 5.13, primer dva taska, pri čemu je task B u fokusu i vrši interakciju sa korisnikom, dok je task A u pozadini sa kompletno očuvanim back stack−om 163
Slika 5.14, slučaj više instanci jedne aktivnosti u okviru back stack−a.....163
Slika 5.15, Prikaz aktivnosti koja preuzima podatke sa korisinčkog interfejsa i ispisuje ih u tekstualnom polju.....168
Slika 5.16, dinamički kreiran korisnički interfejs.....173
Slika 5.17, primer aplikacije koja koristi eksplicitnu nameru za pokretanje druge aktivnosti u okviru iste aplikacije.....180
Slika 6.1, Application not responding dijalog.....181
Slika 6.2, dijagram toka AsyncTask−a.....184
Slika 6.3, Preuzimanje slike pomoću AsyncTask.....190
Slika 6.4, Prikaz dohvaćene liste fakulteta preko AsyncTask−a.....198
Slika 7.1, dijalog traženja dozvola za vreme izvršavanja.....203

Slika 7.2, dijalog traženja dozvola za vreme instalacije.....203
Slika 7.3, dijagram procesa zahtevanja dozvola.....207
Slika 7.4, korisnički interfejs aplikacije koja demonstrira dinamičku dodelu dozvola.....209
Slika 7.5, Rad sa dozvolama, dinamička provera dozvola.....214
Slika 7.6, Rad sa dozvolama, dinamičko traženje dozvola.....215
Slika 7.7, Rad sa dozvolama, dinamička dodela dozvola i provera statusa dozvola za aplikaciju u okviru podešavanja.....215
Slika 7.8, Rad sa dozvolama, dijalog ukoliko korisnik odbije da dodeli dozvole 216
Slika 7.9, dinamička dozvola pristupa eksternom prostoru.....221
Slika 8.1, primer modularnog dizajna upotrebom fragmenata.....223
Slika 8.2, životni ciklus fragmenta dok se njegova aktivnost domaćin izvršava 225
Slika 8.3, ubacivanje fragmenta u aktivnost deklaracijom u XML rasporedu aktivnosti.....228
Slika 8.4, ubacivanje fragmenta programskim putem.....229
Slika 8.5, zamena jednog fragmenta drugim.....231
Slika 8.6, korisnički interfejs aplikacije sa dva fragmenta.....242
Slika 9.1, Room − osnovni dijagram arhitekture i odnosa između komponenti 257
Slika 9.2, Korisnički interfejs aplikacije za čuvanje podataka.....265
Slika 9.3, Prikaz svih unetih redova iz Notes SQLite baze podataka.....279
Slika 10.1, publisher−subscriber šablon.....287
Slika 10.2, prikaz rada BroadcastReceiver aplikacije koja prati obaveštenja o promenama nivoa baterije.....293
Slika 10.3, aplikacija sa prijemnikom obaveštenja o bateriji − ulazak u Low Battery.....297
Slika 10.4, aplikacija sa prijemnikom obaveštenja o bateriji − izlazak iz Low Battery.....297
Slika 10.5, simulacija slanja poruka i poziva ka emulatoru.....302
Slika 10.6, presretanje SMS poruke pomoću BroadcastReceiver komponente 302
Slika 10.7, presretanje poziva pomoću BroadcastReceiver komponente.....303
Slika 10.8, Primer aplikacije koja šalje sopstvena obaveštenja.....306
Slika 11.1, životni ciklus servisa.....309
Slika 11.2, red za čekanje i Looper.....315

Slika 11.3, životni ciklus servisa koji je startovan i vezan u isto vreme.....320
Slika 11.4, prikaz rada servisa koji pokreće muziku u pozadini.....331
Slika 11.5, prikaz rada aplikacije sa IntentService.....335
Slika 11.6, primer izvršavanja vezanog servisa.....340
Slika 11.7, prikaz rada servisa koji obavlja Cezarovu šifru.....349
Slika 12.1, upotreba postojeće aplikacije za kameru upotrebom implicitne namere.....352
Slika 12.2, primer dohvatanja lokacije uređaja.....356
Slika 12.3, Koordinatni sistem koji koriste senzori pokreta na Android uređaju 357
Slika 12.4, senzor rotacije uređaja.....359
Slika 12.5, primer rada akcelerometra.....360
Slika 12.6, kreiranje Google Maps aktivnosti.....361
Slika 12.7, aplikacija koja postavlja marker trenutne lokacije na mapi.....367
Slika 12.8, prikaz markera na mapi sa dodatnim informacijama o lokaciji.....367

Predgovor

Programski jezik Java, kao jedan od najpopularnijih programskih jezika u svetu, svoju popularnost duguje pre svega eleganciji, savremenim konceptima i jednostavnom objektno orijentisanom modelu. Programi napisani u Javi se mogu pronaći svuda, od Android mobilnih uređaja, preko klasičnih desktop aplikacija, pa sve do Internet strana i veb servisa. Zbog sve veće popularnosti mobilnih aplikacija i dostupnosti mobilnih uređaja, jedno od najtraženijih zanimanja danas je upravo razvoj mobilnih aplikacija za Android platformu, što predstavlja fokus ovog udžbenika.

Mobilni operativni sistemi poput Androida su danas dostupni ne samo za mobilne telefone, već i za tablete, kao i čitav niz drugih uređaja, poput pametnih satova i televizora. Samim tim je i tržište na koje se mogu plasirati Android aplikacije značajno povećano. Sa druge strane, mobilne aplikacije imaju veći broj prednosti u odnosu na tradicionalne desktop aplikacije. Ovakav tip aplikacija je uvek uz korisnika, pošto mobilni telefon nosimo uvek sa sobom. Mobilni telefon nudi još dodatne mogućnosti u obliku pristupa Internetu, velikog broja senzora (poput akcelerometra ili kompasa), pristupa mapama, mogućnosti za pozicioniranje preko GPS i kamere, a veliki broj dostupnih aplikacija je besplatan.

Koncept razvoja mobilnih aplikacija za Android u programskom jeziku Java zahteva razumevanje paradigmi na kojima je zasnovan, uključujući dobro poznavanje objektno orijentisanog programiranja (OOP). OOP se bazira na premisi da se problem koji se rešava može modelovati objektima u domenu problema. OOP ima svoje specifične karakteristike, poput nasleđivanja, enkapsulacije i polimorfizma. Programiranje za Android naročito zahteva dobro poznavanje koncepta nasleđivanja, apstraktnih klasa i interfejsa, jer je većina standardnih Android komponenti realizovana upravo na ovaj način, odnosno od programera se zahteva da proširi odgovarajuće standardne Android klase i da implementira već postojeće interfejse. Dodatno, potrebno je imati u vidu i ograničenja na koja ranije nije obraćano previše pažnje. Na primer, veličina ekrana za prikaz je veoma ograničena i drastično je manja od veličine standardnih desktop monitora. Pošto je telefon mobilni uređaj, mora se obratiti pažnja i na ograničen kapacitet baterije i na ograničenu procesorsku moć, koja je višestruko manja od modernih računara.

Programersko okruženje koje se koristi za pisanje Android aplikacija je isključivo Android Studio. U pitanju je okruženje koje je zasnovano na IntelliJ okruženju, i u potpunosti prilagođeno lakšem razvoju mobilnih aplikacija za Android, uz podršku i za Java i za Kotlin. Android Studio, dodatno, značajno olakšava razvoj korisničkog interfejsa, koji se može izvesti prostim

prevlačenjem ponuđenih komponenti na virtuelni ekran. U ovom udžbeniku će čitalac detaljno biti upućen

u korišćenje ovog okruženja za razvoj aplikacija, uz veliki broj konkretnih primera.

Obim i sadržaj udžbenika prilagođeni su nastavnom programu istoimenog predmeta na Fakultetu za informatiku i računarstvo Univerziteta Singidunum u Beogradu. Podrazumeva se da čitaocima knjige ovo nije prvi susret sa programiranjem u Javi, tj. očekuje se poznavanje programskog jezika Java i koncepata OOP.

Tekst je propraćen velikim brojem slika i detaljno objašnjenih primera pomoću kojih su ilustrovani novouvedeni pojmovi. Primeri su odabrani na takav način da budu što jednostavniji za razumevanje, ali sa druge strane i što realističniji i interesantniji, sa krajnjim ciljem da podstaknu čitaoce na dublje samostalno učenje i budu početna tačka za dalje eksperimentisanje. Iako je knjiga pisana kao udžbenik, ambicija autora jeste da ona posluži svima koji se prvi put susreću sa programiranjem mobilnih aplikacija za Android u programskom jeziku Java. Na kraju većine poglavlja data su pitanja koja mogu pomoći u utvrđivanju gradiva. Ova pitanja su i tipična pitanja koja se mogu očekivati na intervjuu za posao Android programera.

Autor se ovom prilikom zahvaljuje asistentu Milanu Tairu, koji je omogućio studentima veći broj primera za vežbu kroz video materijale dostupne na adresi: http://zadatak.singidunum.ac.rs/videos/android/index.php. Iako su video materijali stari nekoliko godina, veliki broj njih je i dalje aktuelan i direktno primenjiv u praksi (uz eventualne manje modifikacije). Neki od primera su poslužili autoru kao inspiracija za pojedine primere koda prikazane u ovom udžbeniku.

Sve primedbe, komentari, preporuke, pohvale i eventualno uočene greške se mogu poslati na adresu mzivkovic@singidunum.ac.rs.

Autor

Beograd, Srbija, 2020.

1. Uvod

Android je operativni sistem za mobilne uređaje koji se zasniva na Linux jezgru, razvijen od strane kompanije Google primarno za uređaje sa ekranom osetljivim na dodir, poput mobilnih telefona i tableta. Korisnički interfejs Androida se uglavnom zasniva na direktnoj manipulaciji putem dodira i različitih pokreta (prevlačenje – engl. *swipe,* zumiranje i odzumiranje – engl. *pinch* itd.) nad objektima koji se nalaze prikazani na ekranu uređaja, uz virtuelnu tastaturu koja se koristi za unos teksta.

Nakon velikog uspeha Android platforme na mobilnim telefonima i tabletima, Google je proširio podršku i na druge uređaje, tako da su danas podržani Android TV za pametne televizore, Android Wear za pametne satove, kao i Android Auto kao podrška za automobile, pri čemu svaki od njih ima specifičan korisnički interfejs. Dodatno, Android je u različitim varijantama danas dostupan i na digitalnim kamerama, konzolama za igre i velikom broju drugih elektronskih uređaja.

Danas se Android nalazi u vlasništvu konzorcijuma Open Handset Alliance, u kome se uz Google nalaze još 83 kompanije. Google je bio predvodnik osnivanja ovog konzorcijuma 2007. godine, zajedno sa kompanijama koje proizvode mobilne uređaje, nekim mobilnim operaterima, proizvođačima čipova, kao i sa kompanijama koje se bave razvojem aplikacija. Istaknuti članovi ovog konzorcijuma, uz naravno Google, su Samsung, LG, HTC, Sony, Motorola, Qualcomm, Texas Instruments, ARM, T-mobile, Vodafone i mnogi drugi. Open Handset Alliance je zadužen za razvoj otvorenih standarda za mobilne uređaje, a vodeći (engl. *flagship*) softver ovog konzorcijuma je upravo Android, koji se zasniva na licenci otvorenog koda. U cilju promovisanja jedinstvene platforme, svim članicama Open Handset Alliance je zabranjeno da proizvode mobilne uređaje na kojima se nalaze nekompatibilne verzije Android sistema. Kompatiblinost se postiže time što se softver razdvaja od hardvera na kome se izvršava, čime se omogućava velikom broju heterogenih uređaja da izvršavaju istu aplikaciju. Prvi komercijalno dostupni model telefona sa Android operativnim sistemom bio je HTC Dream, koji je bio poznat i pod nazivom T- Mobile G1, koji se na tržištu pojavio krajem 2008. godine.

Svaka od verzija Androida ima dodeljen broj, koji se naziva API nivo (engl. *Application Programming Interface - API level*). Ova oznaka služi da označi koja verzija Android API (uključuje veliki broj klasa i biblioteka) je dostupna programerima pri razvoju aplikacije. Osim ove oznake, glavne (engl. *major*) verzije imaju i kodno ime, poput Lollypop, Marshmallow, Nougat, Oreo itd. Specifično je da su sva kodna imena zasnovana na engleskim nazivima za različite slatkiše i idu abecednim redosledom. Ova praksa završena je 2019.

godine, objavljivanjem Android 10 platforme, uz najavu da će se i sve buduća imena zasnivati na brojevima. Moguće je da više verzija androida ima isti API nivo, poput verzija Gingerbread 2.3.3, 2.3.4, 2.3.5, 2.3.6 i 2.3.7, koje sve imaju isti API nivo 10. Praktično gledano, time se označava da sve pomenute Gingerbread verzije nisu imale promene samog API-ja koji je dostupan i vidljiv programeru, već da su u pitanju samo ispravke uočenih bagova i određena unapređenja same platforme.

1.1. Istorijat

Razvoj Androida započeo je 2003. godine, pod okriljem kompanije Android Inc. Ovu kompaniju je 2005. godine preuzeo Google, a od 2007. godine Android se nalazi u okviru konzorcijuma Open Handset Alliance. Prva beta verzija objavljena je 5. novembra 2007. godine, dok je prvi SDK (engl. *software development kit*) objavljen nedelju dana kasnije, 12. novembra 2007. godine. Usledilo je još nekoliko beta verzija samog SDK, koja su se zasnivala isključivo na softverskoj emulaciji, pošto još uvek nije postojao nijedan fizički uređaj kako bi se testirao sam operativni sistem. Prva zvanična i javno dostupna verzija Androida 1.0 usledila je u oktobru 2008. godine sa objavom prvog komercijalno dostupnog Android telefona – HTC Dream (slika 1.1). Android 1.0 i 1.1 nisu imali dodeljena specifična kodna imena. Interno su bili poznati pod nazivima Astro Boy i Bender, ali ta imena nisu izašla u javnost, tako da su zvanično ostali priznati pod jednostavnim imenima 1.0 i 1.1.

Slika 1.1, prvi komercijalno dostupni Android telefon HTC Dream (slika preuzeta sa https://android.appsapk.com/product/htc-dream/)

1.1.1. Prve verzije

Android verzija 1.0 (API nivo 1) je sadržavala veći broj tada veoma naprednih funkcionalnosti:

- Android Market aplikacija koja se koristila za download i update drugih aplikacija.
- Veb čitač za prikaz veb stranica, sa podržanom opcijom za zumiranje i prikazom više otvorenih stranica istovremeno kroz kartice.
- Kamera – uz ograničenja da nije bilo moguće promeniti rezoluciju i kvalitet slika.
- Folderi koji su omogućili grupisanje ikonica aplikacija u jedinstveni folder na glavnom ekranu.
- Pristup mejl serverima putem protokola POP3, IMAP4 i SMTP.
- Sinhronizacija Gmail naloga sa Gmail aplikacijom na telefonu.
- Google kontakti sinhronizovani sa People aplikacijom.
- Google kalendar sinhronizovan sa Calendar aplikacijom.
- Google Maps za prikaz mapa i satelitskih snimaka, uz podršku za navigaciju preko GPS (Global Positioning System).
- Google Sync za upravljanje sinhronizacijom aplikacija Gmail, People i Calendar.
- Google Search za pretragu Interneta i aplikacija na telefonu, kontakata, kalendara i slično.
- Google Talk za komunikaciju preko poruka.
- Instant poruke, tekstualne poruke (SMS) i multimedijalne poruke (MMS).
- Media Player za upravljanje i prikaz multimedijalnim fajlovima (uz nedostatak video i stereo Bluetooth podrške).
- Notifikacije koje se prikazuju u statusnoj liniji, sa različitim opcijama za podešavanje zvučnih signala telefona, LED svetala i vibracije.
- Voice Dialer za usmeno iniciranje poziva, bez ukucavanja imena ili broja.
- Wallpaper za postavljanje različitih pozadinskih slika ili fotografija iza ikonica i vidžeta na glavnom ekranu (Home screen).
- Inkorporirani YouTube plejer.
- Podrška za Wi-Fi i Bluetooth.
- Dodatne aplikacije, poput Alarm Clock, Calculator, Dialer, Gallery i Settings.

Sledeća verzija, Android 1.1 (API nivo 2), objavljena je u februaru 2009 za telefon HTC Dream. Ova verzija je ispravila neke uočene bagove i donela nekoliko novih funkcionalnosti, od kojih su najbitnije:

- Prilikom pretrage za lokalnim poslovnicama (poput restorana) na mapama, moguće je videti i detalje i komentare korisnika.
- Mogućnost čuvanja priloga (engl. *attachments*) u okviru poruka.
- Mogućnost sakrivanja/prikazivanja tastature za vreme poziva preko spikerfona.
- Podrška za *marquee* (nešto slično kajronu na televiziji, tekst koji skroluje preko ekrana) u rasporedu komponenti (engl. *layout*).

1.1.2. Android 1.5 Cupcake

U toku 2009. godine pojavili su se novi Android mobilni telefoni, kao i nekoliko verzija samog Androida. Prva od ovih verzija je Android 1.5 (API nivo 3), koji je objavljen u aprilu 2009, koja je prva imala zvanično kodno ime bazirano na slatkišu – Cupcake. Verzija je donela nekoliko novih funkcionalnosti, kao i izmena korisničkog interfejsa, čiji je izgled prikazan na

slici 1.2.

Slika 1.2, Android 1.5 Cupcake glavni ekran
(slika preuzeta sa: https://en.wikipedia.org/wiki/Android_version_history)

Najbitnije nove funkcionalnosti su:

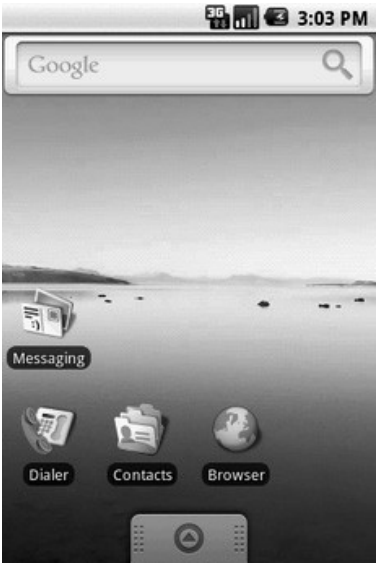
- Podrška za virtuelne tastature razvijane od treće strane (engl. *third party*), sa predikcijom teksta i rečnikom za korisnički definisane pojmove
- Podrška za vidžete (engl. *widgets* – male aplikacije sa minijaturnim prikazom, koje se mogu ugnjezditi u druge aplikacije, poput glavnog ekrana, i koje se mogu periodično ažurirati).

- Snimanje i plejbek video materijala u MPEG-4 i 3GP formatima.
- Bluetooth - automatsko uparivanje i podrška za stereo.

- Podrška za copy/paste u veb čitaču.
- Slike za omiljene kontakte.
- Animirane tranzicije između ekrana.
- Auto-rotacija.
- Upload video materijala na YouTube i slika na Picasa.

1.1.3. Android 1.6 Donut

Sledeća verzija Androida, koja je takođe izašla 2009. godine, u septembru, je bila verzija 1.6, poznata pod kodnim imenom Donut. Izgled korisničkog interfejsa prikazan je na slici 1.3.



Slika 1.3, Android 1.6 Donut glavni ekran
(slika preuzeta sa: https://en.wikipedia.org/wiki/Android_version_history)

Uvedene su brojne nove funkcionalnosti, od kojih su najbitnije:

- Tekstualna i glasovna pretraga je poboljšana tako da uključuje istoriju pretraživanja, obeležene stranice (engl. *bookmarks*) i veb.
- Mogućnost da programeri uključe svoj sadržaj u rezultate pretrage.
- Lakša pretraga aplikacija na Android Market-u, sa dodatim skrinšotovima aplikacija.
- Galerija i kamera su potpuno integrisani, sa bržim pristupom kameri.
- Mogućnost obeležavanja i brisanja više slika odjednom.
- Podrška za text-to-speech.

- Podrška za telekomunikacione tehnologije CDMA (engl. *Code-division multiple access*), EVDO (engl. *Evolution-Data Optimized*), VPN (engl. *Virtual Private Network*). Jedna od mreža koja koristi CDMA je Verizon u Sjedinjenim Američkim Državama, čime je drastično uvećano tržište.
- Proširena podrška za pokrete i gestove prstima.

1.1.4. Android 2.0 Eclair

Još jedna verzija Androida je puštena u javnost 2009. godine, krajem oktobra objavljen je Android 2.0 Eclair (API nivo 5). Izgled korisničkog interfejsa ove verzije prikazan je na slici 1.4. Već u decembru iste godine izašla je verzija Android 2.0.1 Eclair (API nivo 6), a u januaru 2010 i verzija Android 2.1 Eclair (API nivo 7). Ove dve verzije su sadržavale samo manje ispravke bagova i minorne promene API-ja. Eclair je uveo sledeće nove funkcionalnosti:

- Proširenje sinhronizacije naloga, tako da se korisnicima dozvoli da dodaju više naloga na uređaj koji se mogu sinhronizovati sa mejlom i kontaktima.
- Podrška za Microsoft Exchange, sa kombinovanim sandučetom gde se mogu videti mejlovi sa više naloga u okviru jedne strane.
- Podrška za Bluetooth 2.1.
- Dodir na fotografiju kontakta otvara opcije za iniciranje poziva, SMS ili mejl poruke.
- Brojna poboljšanja kamere, uključujući blic, digitalno zumiranje, makro fokus, različite kolor efekte i slično.
- Ubrzan rad virtuelne tastature, sa pametnijim imenikom koji uči reči koje korisnik upotrebljava, i uključuje imena kontakata u predloge.
- Osvežen korisnički interfejs veb čitača, uz dodatu podršku za HTML5 standard i za zumiranje dvostrukim dodirom.
- Poboljšane funkcionalnosti kalendara i Google Maps (glasovna navigacija).
- Optimizovana brzina sistema i podrška za veći broj veličina ekrana i različitih rezolucija, sa boljim odnosom kontrasta.
- Podrška za rukovanje multi-touch događajima.
- Podrška za žive pozadine kroz animaciju pozadine glavnog ekrana.
- Podrška za speech-to-text.

Slika 1.4, Android 2.0 Eclair poboljšana navigacija i funkcija speech-to-text (slika preuzeta sa: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html)

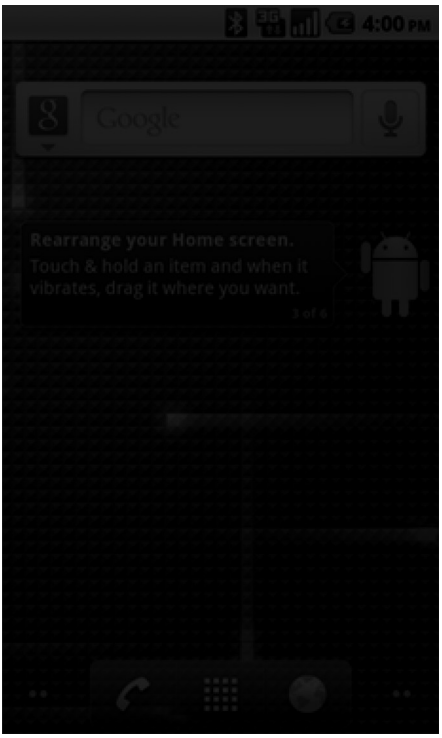
1.1.5. Android 2.2 Froyo

U maju 2010. godine usledila je verzija Android 2.2 Froyo (API nivo 8). Froyo je inače skraćeno od zamrznutog jogurta (engl. *Frozen Yogurt*). Izgled korisničkog interfejsa ove verzije prikazan je na slici 1.5. Ova verzija je unela brojna poboljšanja i optimizacije, među kojima su najbitnije:

- Brojne optimizacije brzine, korišćenja memorije i performansi.
- Dodatna ubrzanja aplikacija kroz JIT kompilaciju (engl. *Just in Time*).
- Podrška za Android Cloud to Device Messaging servis, čime su omogućene push notifikacije.
- Deljenje mreže - USB *tethering* i Wi-Fi hotspot funkcionalnosti.
- Opcija za isključivanje prenosa podataka preko mobilne mreže.
- Ažurirana Android Market aplikacija, sa opcijom za automatsko ažuriranje aplikacija.
- Brz prelaz između različitih jezika na virtuelnoj tastaturi.

- Podrška za numeričke i alfanumeričke šifre.
- Podrška za Adobe Flash
- Veb čitač prikazuje sve frejmove animiranog GIF-a, umesto samo prvog

Froyo je imao još tri verzije u toku 2011. godine, 2.2.1, 2.2.2 i 2.2.3, koje su sve bile samo ispravke bagova, dodatna poboljšanja performansi i sigurnosne

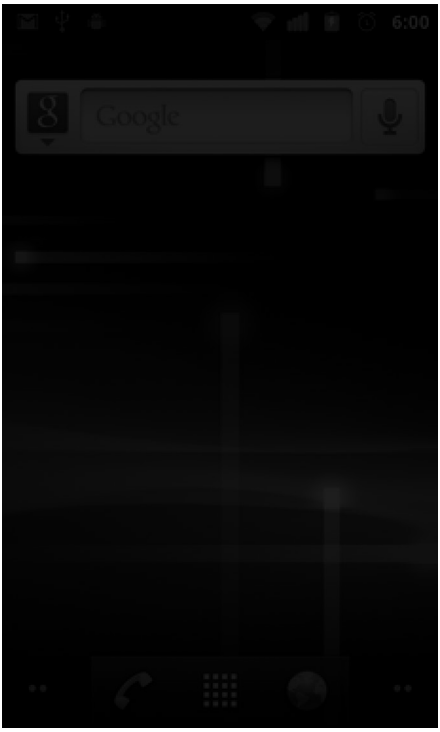


zakrpe.

Slika 1.5, Android 2.2 Froyo glavni ekran
(slika preuzeta sa: https://en.wikipedia.org/wiki/Android_version_history)

1.1.6. Android 2.3 Gingerbread

U decembru 2010. godine, usledila je i nova glavna verzija Androida, Android 2.3 Gingerbread (API nivo 9). Izgled korisničkog interfejsa ove verzije prikazan je na slici 1.6. Ova verzija počela je da uvodi i prvi pravi vizuelni identitet Androida. Svetlo zelena boja je oduvek bila boja Androidove maskote – robota, a sa Gingerbread verzijom, postala je integralni deo samog izgleda operativnog sistema. Crna i zelena boja su bile prisutne i vidljive u celom korisničkom interfejsu, čime je počelo utabavanje prepoznatljivog dizajna.



Slika 1.6, Android 2.3 Gingerbread glavni ekran
(slika preuzeta sa: https://developer.android.com/about/versions/android-2.3-highlights)

Osim prepoznatiljvog interfejsa, ova verzija je takođe uvela veći broj novina i poboljšanja, među kojima su najbitniji:

- Podrška za veoma velike ekrane i rezolucije (WXGA i veće).
- Nativna podrška za SIP VoIP (engl. *voice over IP*)
- Brži i intuitivniji unos teksta preko virtualne tastature, sa poboljšanom preciznošću, boljim sugerisanjem teksta i boljim glasovnim unosom.
- Poboljšana copy/paste opcija, koja dozvoljava korisnicima da odaberu i kopiraju reč dodirom i držanjem.
- Podrška za NFC (engl. *Near Field Communication*), čime korisnici mogu da očitaju NFC tagove koji su embedovani u okviru različitih postera ili reklama.
- Bolji audio efekti, naročito pri upotrebi slušalica.
- Novi Download Manager, koji omogućava korisnicima lak pristup bilo kom fajlu koji je preuzet putem veb čitača, mejla ili neke druge aplikacije.

- Podrška za više kamera na uređaju, poput prednje kamere ukoliko je dostupna na samom uređaju.
- Podrška za nove video/audio formate.
- Poboljšani menadžment energije, koji aktivnije prati aplikacije koje predugo drže uređaj budnim i time troše mnogo energije.
- Konkurentno sakupljanje đubreta (engl. *Garbage Collection*) za poboljšanje performansi.
- Veliki broj audio i grafičkih poboljšanja za bolju podršku mobilnim video igrama.

Nakon ove verzije, usledilo je još nekoliko Gingerbread verzija. Verzije 2.3.1 i 2.3.2 (API nivo 9) su se uglavnom odnosile na poboljšanja i ispravke bagova za model Nexus S. Nexus S je bio telefon koji su zajedno razvili Google i Samsung, i bio je prvi uređaj koji je koristio Gingerbread 2.3, kao i prvi Android uređaj sa softverskom i hardverskom podrškom za NFC. U februaru 2011. godine izašla je i verzija 2.3.3, koja je takođe nosila oznaku Gingerbread, ali sa novim API nivoom 10. Od februara do septembra 2011. godine objavljeno je 5 verzija, 2.3.3
– 2.3.7, koje su uglavnom sadržavale ispravke uočenih grešaka (između ostalog spontano resetovanje nekih modela, bagovi sa Bluetooth i Wi-Fi, kao i bag u glasovnoj pretrazi), zatim određenja poboljšanja (mrežne performanse, bolja Gmail aplikacija, dodate animacije pri skrolovanju lista, poboljšanja kamere), kao i dodavanje podrške za Google Walet sa verzijom 2.3.7.

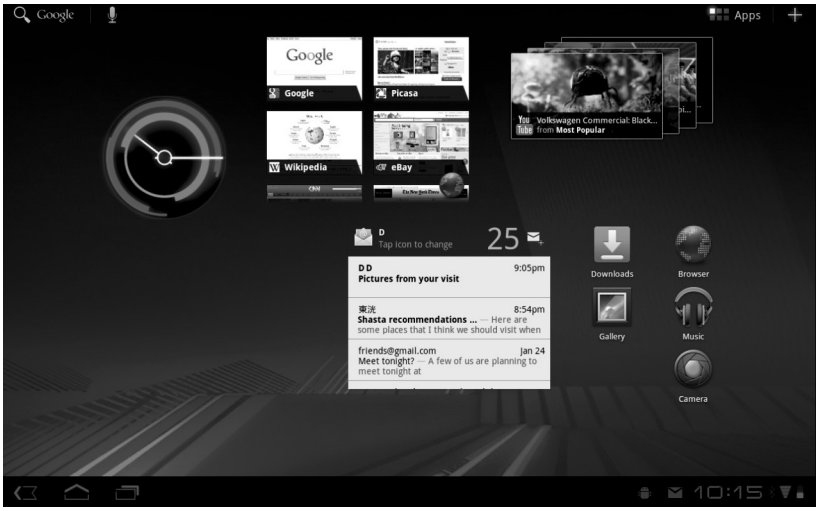
1.1.7. Android 3.0 Honeycomb

U februaru 2011 objavljena je verzija Android 3.0 Honeycomb (API nivo 11), koja je bila namenjena i optimizovana isključivo za uređaje sa većim ekranima - tablete. Prvi tablet koji je podržavao ovu verziju Androida bio je Motorola Xoom. Nakon verzije 3.0, iste godine usledile su verzije 3.1 (API nivo 12) i 3.2 (API nivo 13). Uveden je novi, tzv. hologramski dizajn, kao što je prikazano na slici
1.7. Ovakav dizajn je omogućio maksimalno iskorišćenje velikog ekrana tableta. Iako sam koncept interfejsa specifičnog isključivo za tablete nije potrajao dugo, veliki broj ideja koje je uveo Honeycomb su postavile temelje za Android kakav danas poznajemo, poput softverskih dugmića za navigaciju na samom ekranu i koncepta prikaza nedavno korišćenih aplikacija u obliku kartica. Među ostalim novinama koje su unete sa ovom verzijom, izdvajaju se:

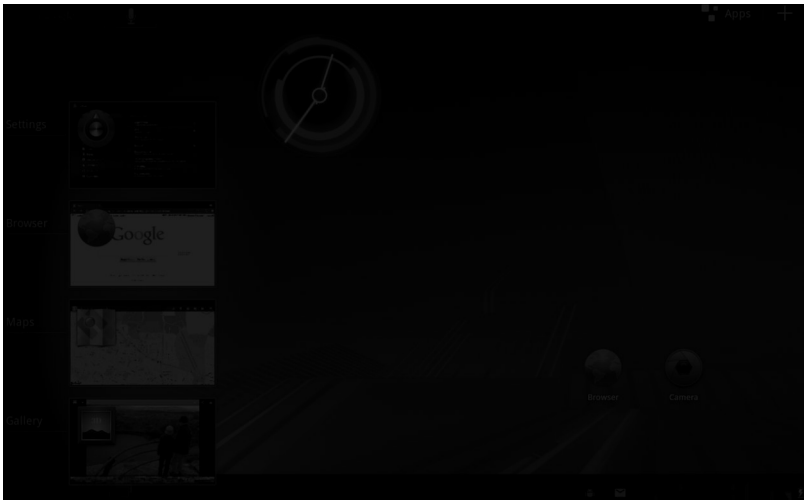
- Dodata sistemska linija (engl. *System bar*) sa brzim pristupom notifikacijama i statusu, kao i softverski navigacioni dugmići na dnu ekrana.

- Dodata akciona linija (engl. *Action bar*) sa pristupom kontekstualnim opcijama, navigaciji i vidžetima.

- Pojednostavljen multitasking – dodir na nedavno korišćene aplikacije (engl. *Recent Apps*) omogućava korisniku da vidi sličice (kartice) taskova koji se trenutno izvršavaju, i da brzo skače sa jedne aplikacije na drugu, kao što je prikazano na slici 1.8.
- Podrška za procesore sa više jezgara.
- Dvopanelni korisnički interfejsi za kontakte i mejlove, za lakšu organizaciju i prikaz kontakata i organizaciju mejlova.



Slika 1.7, Android 3.0 Honeycomb glavni ekran
(izvor: https://developer.android.com/about/versions/android-3.0-highlights)



Slika 1.8, Android 3.0 Honeycomb Recent apps
(izvor: https://developer.android.com/about/versions/android-3.0-highlights)

1.1.8. Android 4.0 Ice Cream Sandwich

U oktobru 2011. godine na tržište je izbačena još jedna glavna verzija - Android 4.0 Ice Cream Sandwich (API nivo 14). Smatra se da je prethodna verzija, Honeycomb, služila kao most za prelazak između starog i novog, a da je Ice Cream Sandwich zvanično uveo platformu u eru modernog dizajna. Ice Cream Sandwich je iskoristio i poboljšao sve vizuelne koncepte uvedene sa verzijom Honeycomb (softverski dugmići za navigaciju i pregled nedavnih aplikacija), i ujedinio tablete i telefone pod jedinstven korisnički interfejs, koji je prikazan na slici 1.9.

Slika 1.9, Android 4.0 Ice Cream Sandwich glavni ekran i Recent Apps (slika preuzeta sa: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html)

Tokom 2011. i 2012. godine objavljeno je još nekoliko verzija Ice Cream Sandwich. Verzije 4.0.1 i 4.0.2 su delile isti API nivo 14 kao i glavna verzija, dok su verzije 4.0.3 i 4.0.4 objavljene pod API nivoom 15. Naknadne verzije su uglavnom sadržavale ispravke bagova i manja poboljšanja. Osim izgleda korisničkog interfejsa, nove funkcionalnosti koje su uvedene sa Ice Cream Sandwich verzijom su brojne, a među najznačajnije spadaju:

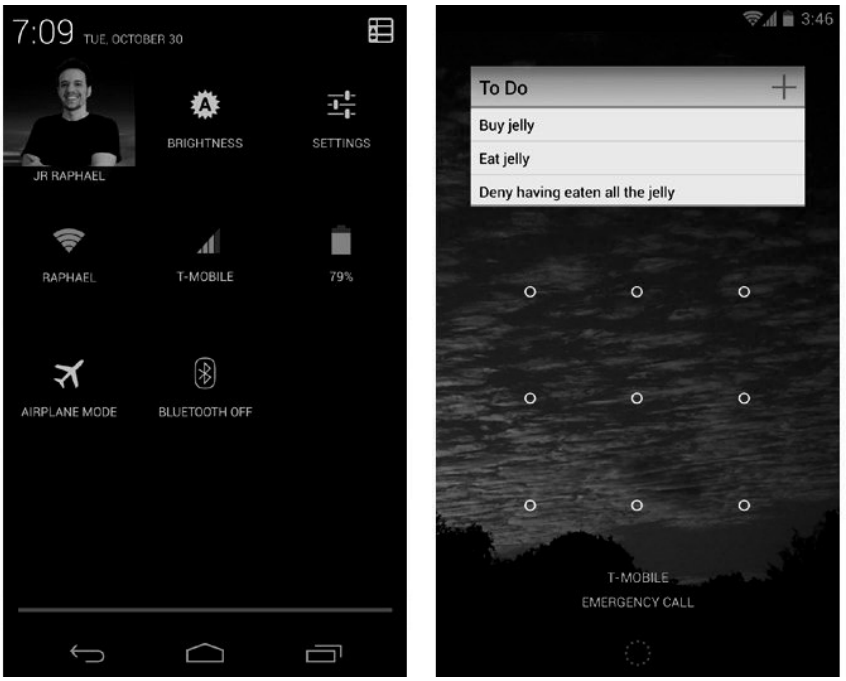
- Integrisana podrška za skrinšot (istovremeno držanje hardverskih dugmića *Power* i *Volume-down*).
- Poboljšano zumiranje pokretima prstiju u nekim aplikacijama poput kalendara.
- Poboljšan copy/paste.
- Poboljšano prepoznavanje lica, uvedena opcija otključavanja telefona sa Face Unlock.
- Dodata sekcija Data Usage u okviru podešavanja, koja upozorava korisnika kada se približi određenom limitu transfera podataka preko mobilne mreže, i onemogućava prenos podataka preko mobilne mreže kada se ovaj limit premaši.
- Pokretom prevlačenja (engl. *swipe*) se može ukloniti aplikacija iz liste nedavnih aplikacija ili odbaciti notifikacija.
- People aplikacija je poboljšana integracijom socijalnih mreža, statusnim porukama i slikama visoke rezolucije.
- Poboljšana kamera, dodat panorama režim rada, kao i mogućnost zumiranja za vreme snimanja video snimka.
- Glatke rotacije ekrana.

1.1.9. Android 4.1 Jelly Bean

U julu 2012. godine, na tržište je stigla verzija Android 4.1 Jelly Bean (API nivo 16). Tokom 2012. i 2013. godine izašle su još dve verzije Jelly Bean: 4.2 (API nivo 17) i 4.3 (API nivo 18). Ova verzija je dodatno nadgradila i optimizovala prethodnu verziju, čime je Android postao mnogo privlačniji prosečnim korisnicima. Izgled korisničkog interfejsa ove verzije prikazan je na slici 1.10. Velika poboljšanja u performansama i brzini rada Androida obezbeđena su, između ostalog, smanjenim kašnjenjem detekcije dodira, ubrzavanjem procesora i hardverski ubrzanim 2D renderovanjem. Notifikacije su sada postale proširive i interaktivne, poboljšana je i podrška za glasovnu pretragu i uveden je napredniji sistem za prikaz rezultata pretrage. Osim navedenih, neke od najznačajnijih novih funkcionalnosti koje je uveo Jelly Bean su:

- Mogućnost za ukidanje notifikacija na nivou pojedinačnih aplikacija.
- Prečice i vidžeti se mogu aranžirati i promeniti veličinu kako bi se dodali novi na glavnom ekranu.
- Dodatno poboljšan softver za kameru.
- Višekanalni i USB audio
- Brza podešavanja (engl. *Quick Settings*).
- Nova Clock aplikacija, sa ubačenim svetskim satom, štopericom i tajmerom.

- Podrška za Bluetooth LE (engl. *Low energy*) i opšte gledano poboljšan Bluetooth stek.
- Poboljšana grafika za igre uvođenjem podrške za OpenGL ES 3.0.
- Omogućeno više korisničkih naloga.
- Podrška za jezike koji se pišu sa desna na levo.
- Nativna podrška za emoji.
- Veći broj sigurnosnih ispravki.
- Poboljšanje zaključanog ekrana, uključujući mogućnost da se putem poteza prevlačenja prstom direktno ode u aplikaciju kamere.



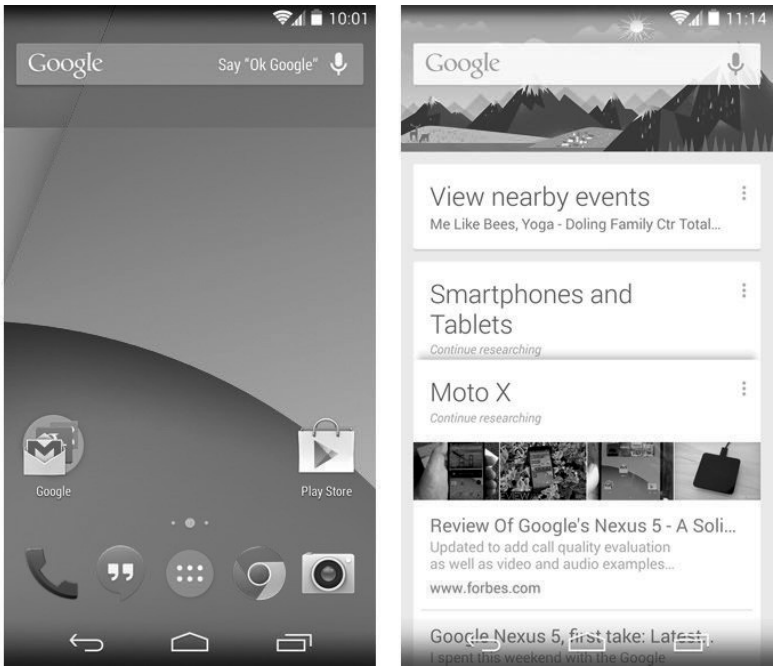
Slika 1.10, Android 4.1 Jelly Bean korisnički interfejs
(slika preuzeta sa: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html)

1.1.10. Android 4.4 KitKat

Sledeća verzija Androida, 4.4 KitKat (API nivo 19), zvanično je izašla na tržište u septembru 2013. Ova verzija je označila kraj "mračne" ere, pošto je pretežno crna boja prisutna u verziji Gingerbread, odnosno pretežno tamno plava koju je uveo Honeycomb zamenjena svetlijim i neutralnijim pozadinama, sa transparentnom statusnom linijom i pretežno svetlim i belim ikonicama. Korisnički interfejs ove verzije Androida prikazan je na slici 1.11. KitKat je

dizajniran da se izvršava brzo i glatko na većem broju uređaja nego ikada pre, računajući veliki broj niskobudžetnih modela telefona sa svega 512MB RAM. Veliki broj optimizacija pomogao je u štednji raspoložive memorije, poput optimizacije Dalvik JIT keša, KSM (engl. *kernel samepage merging*) i prelaska na zRAM. Osim glavne verzije 4.4, do sredine 2014. godine na tržište je izašlo još četiri verzije KitKat-a, označenih sa 4.4.1 – 4.4.4, koje su uglavnom bile ispravke bagova, dodatne optimizacije, kao i poboljšanje sigurnosti. KitKat je uneo brojne novine, a najbitnije nove funkcionalnosti su, između ostalog:

- Aplikacije u imerzivnom modu (sakrivanje navigacije i statusne linije, aplikacija preuzima ceo ekran).
- Restrikcije aplikacijama prilikom pristupa eksternoj memoriji (osim sopstvenih direktorijuma).
- Opcija za bežično štampanje.
- Dodatne opcije sa senzorima, poput detekcije i brojanja koraka korisnika.
- Umesto Dalvik virtuelne mašine započeto uvođenje Android Runtime (ART).
- Aplikacijama razvijenim od trećih strana onemogućen pristup statistici korišćenja baterije.

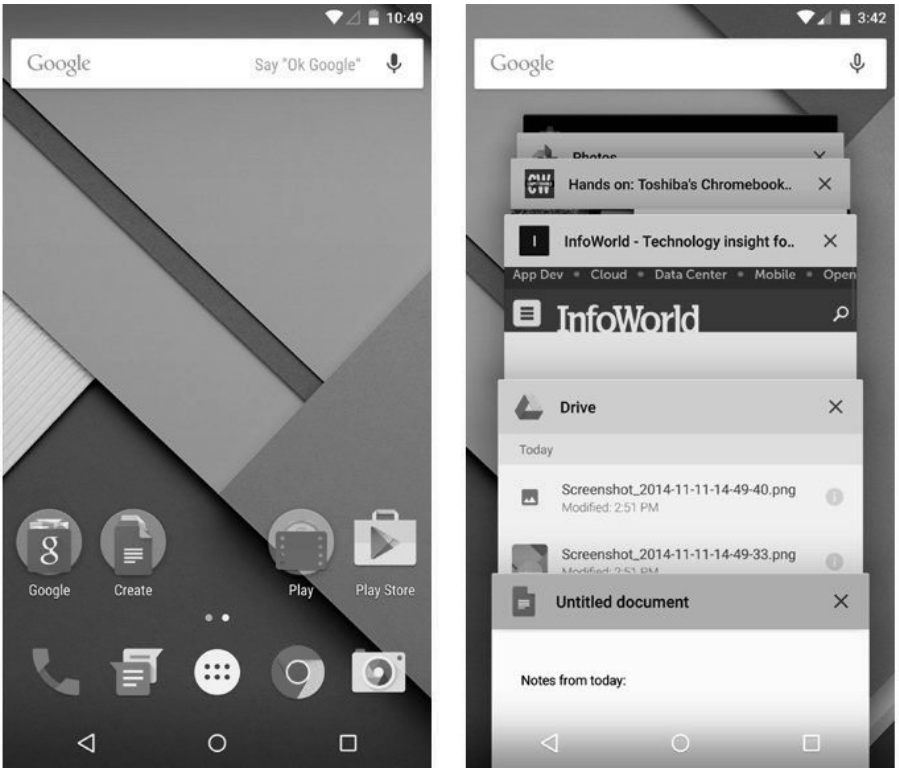


Slika 1.11, Android 4.4 KitKat korisnički interfejs
(slika preuzeta sa: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html)

Osim glavne Android 4.4 verzije, 2014. godine na tržište je puštena i verzija Android 4.4W KitKat, koja je bila isključivo namenjena Android Wear uređajima, pretežno pametnim satovima.

1.1.11. Android 5.0 Lollipop

U junu 2014. godine, na tržište je izašla sledeća glavna verzija, Android 5.0 Lollipop (API nivo 21). Lollipop je uveo i danas prisutni standard materijalnog dizajna, koji je doneo novi izgled u sve delove Androida i aplikacije. Korisnički interfejs ove verzije Androida prikazan je na slici 1.12.



Slika 1.12, Android 5.0 Lollipop korisnički interfejs i materijalni dizajn (slika preuzeta sa: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html)

Osim redizajniranog korisničkog interfejsa, poboljšane su i notifikacije, kojima se može pristupiti direktno sa zaključanog ekrana, a mogu biti prikazane i u gornjem baneru. Google je u ovoj verziji zvanično zamenio Dalvik virtuelnu mašinu sa Android Runtime, kako bi se dobile još bolje performanse i dodatno

optimizovala potrošnja baterije. Osim navedenih, najbitnije nove funkcionalnosti, između ostalog, uključuju:

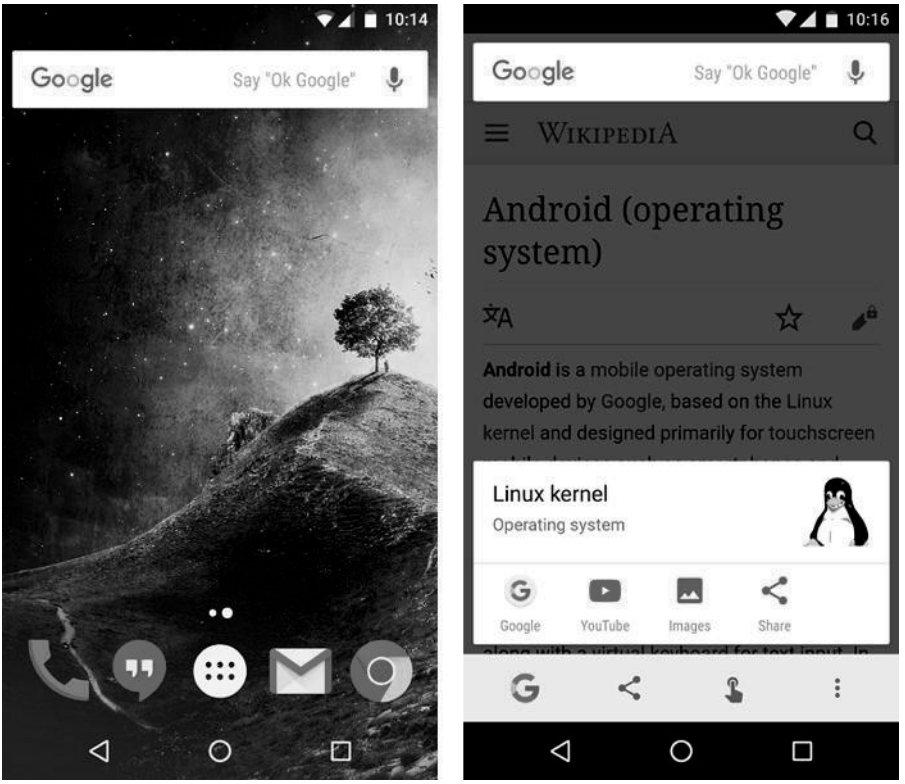
- Podrška za 64-bitne procesore.
- Vektorski grafički elementi (engl. *drawables*), koji se skaliraju bez gubitka kvaliteta.
- Projekat Volta, sa ciljem poboljšanja korišćenja baterije.
- Audio ulaz i izlaz kroz USB uređaje.
- Nedavno korišćene aplikacije su upamćene i posle restarta uređaja.
- Aplikacije razvijene od trećih strana ponovo dobijaju pristup podacima na SD kartici.
- Ubačena aplikacija Flashlight, koja koristi fleš kamere za osvetljavanje.

Osim glavne verzije 5.0, u toku decembra 2014. godine objavljene su još dve verzije Lollipop sistema, 5.0.1 i 5.0.2, sa istim API nivoom 21, uglavnom sa ispravkama uočenih bagova. U martu 2015. godine, izašao je Android 5.1 Lollipop sa API nivoom 22, koji je doneo nekoliko novina, poput zvanične podrške za više SIM kartica u jednom telefonu, dodatne zaštite u slučaju gubitka uređaja i prenosa glasa u visokoj definiciji između uređaja koji su kompatibilni sa 4G LTE. Nešto kasnije, u aprilu 2015. godine, verzija 5.1.1 je uglavnom ispravila uočene bagove.

1.1.12. Android 6.0 Marshmallow

U oktobru 2015 objavljena je nova glavna verzija Android sistema, 6.0 Marshmallow (API nivo 23). Ova verzija je započela trend da postoji jedna glavna verzija Androida godišnje, koja dobija svoj novi ceo broj (u ovom slučaju 6.0). Korisnički interfejs ove verzije Androida prikazan je na slici 1.13.

Jedna od glavnih novina koju je uveo Marshmallow je novi model dozvola (privilegija) sa mnogo većom granulacijom, gde korisnici dodeljuju dozvole za vreme izvršavanja aplikacije. Na ovaj način se korisnicima daje veća kontrola nad dozvolama, a olakšava proces instalacije aplikacija i njihovog ažuriranja. Korisnici u bilo kom trenutku mogu da daju ili povuku dozvole za instalirane aplikacije. Uvedene su i nove optimizacije za štednju energije kroz dva nova režima rada uređaja, Doze i App Standby. Još jedna od bitnih izmena je podrška za čitanje otiska prsta. U decembru 2015, objavljena je i verzija Marshmallow 6.0.1 sa sitnijim unapređenjima.



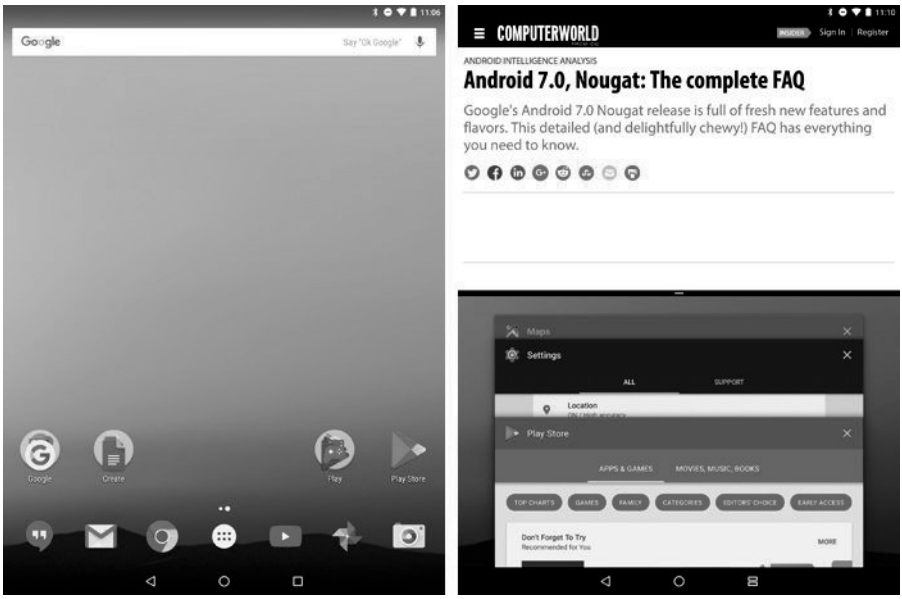
Slika 1.13, Android 6.0 Marshmallow korisnički interfejs
(slika preuzeta sa: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html)

Osim već navedenih novih funkcionalnosti, Marshmallow je uveo veći broj novina, od kojih su najbitnije:

- Direct Share opcija.
- Automatski bekap aplikacija.
- 4k prikaz za aplikacije.
- Eksperimentalna podrška za više prozora.
- Unapređenja USB
- Unicode 7.0 i 8.0 emoji.
- Dvostruki pritisak na Power dugme otvara kameru.

1.1.13. Android 7.0 Nougat

Sledeća glavna verzija Androida izašla je na tržište u avgustu 2016, pod oznakom Android 7.0 Nougat (API nivo 24). Nešto kasnije u toku iste godine, izašla je i verija 7.1 Nougat (API nivo 25), kao i dve manje verzije 7.1.1 i 7.1.2. Glavna nova funkcionalnost koja je vidljiva korisnicima je podrška za podeljeni ekran (engl. *split screen*). Ova funkcionalnost, uz izgled korisničkog interfejsa, prikazana je na slici 1.14. Nougat je takođe uveo nova poboljšanja i optimizacije sa ciljem štednje energije i produžavanja životnog veka baterije, kao i smanjenje potrošnje RAM-a. Režim rada Doze, koji je uveden u Android 6.0, dodatno je poboljšan.



Slika 1.14, Android 7.0 Nougat korisnički interfejs i nova opcija podeljenog ekrana
(slika preuzeta sa: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html)

Osim ovih najbitnijih izmena, druge funkcionalnosti koje je Nougat uveo su:

- Podrška za enkripciju fajlova.
- Novi Unicode 9.0 emoji.
- Dodata opcija Clear All u listi nedavno korišćenih aplikacija.
- Daydream platforma virtuelne stvarnosti (VR interfejs).
- Poboljšanja pregledača fajlova.
- Dodatne opcije u podešavanjima.

- Novi JIT kompajler, koji ubrzava instalacije aplikacija, a smanjuje veličinu kompajliranog koda.
- Slika u slici – podrška za Android TV.
- Vulkan 3D API za renderovanje.
- Poboljšane performanse ekrana osetljivog na dodir
- Podrška za tastaturu sa ubacivanjem slika
- Google asistent

1.1.14. Android 8.0 Oreo

Osma glavna verzija Android sistema je puštena na tržište u avgustu 2017. godine, pod nazivom Android 8.0 Oreo (API nivo 26). U decembru 2017. godine, puštena je još jedna manja verzija, Android 8.1.0 Oreo (API nivo 27). Jedna od najvažnijih novih opcija je specijalan oblik režima rada sa više prozora pod nazivom slika u slici (Picture-in-picture, PIP), koji je prikazan na slici 1.15. Ovaj režim rada je prvenstveno bio dostupan samo na Android TV uređajima, ali je sa Androidom 8.0 postao dostupan na svim Android uređajima. Još jedna od novih funkcionalnosti je mogućnost da se odlože (engl. *snooze*) notifikacije, kao i posebni kanali za notifikacije putem kojih je moguće podesiti na koji način želimo da nas neka aplikacija obavesti. Još jedna od značajnijih izmena je restrukturirana sekcija podešavanja, gde su podsekcije grupisane tako da slične opcije budu zajedno. Dodat je i veći broj novih API-ja, od kojih se izdvaja API za neuronske mreže. Osim navedenih, dodat je veći broj novih funkcionalnosti, od kojih su najznačajnije:

- Pojačana sigurnost kroz Google Play Protect.
- Wi-Fi asistent.
- Automatska aktivacija isključenog WiFi ukoliko se prepozna poznata mreža.
- Adaptivne ikonice.
- Autofill opcija na nivou celog sistema.
- Podrška za više displeja.
- Optimizacija aplikacija koje su u pozadini.
- Dva puta brže podizanje sistema u poređenju sa varijantom Nougat (prema Google, testirano na njihovom Pixel uređaju).
- Vizuelne izmene za Power Off i Restart opcije, sa novim ekranom.
- Vizuelna izmena Toast poruka.

Slika 1.15, Android 8.0 Oreo korisnički interfejs i nova opcija slike u slici (izvor: https://developer.android.com/about/versions/oreo/android-8.0)

1.1.15. Android 9.0 Pie

Deveta glavna verzija Androida, pod nazivom Android 9.0 Pie (API nivo 28), puštena je na tržište u avgustu 2018. Jedna od najinteresantnijih novih opcija je podrška za pozicioniranje u zatvorenom prostoru uz pomoć Wi-Fi RTT (*Wi-Fi Round Trip Time*), koja je prikazana na slici 1.16. Upotrebom RTT API-ja aplikacije mogu da izmere udaljenost do obližnjih Wi-Fi pristupnih tačka (koje imaju opciju RTT) i da na taj način odrede lokaciju uređaja (ovo zahteva da su servis lokacije i Wi-Fi skeniranje uključeni u podešavanjima sistema). Ukoliko uređaj uspe da izmeri udaljenost do tri ili više pristupnih tačaka, pomoću multilateracije moguće je proceniti poziciju uređaja sa tipičnom preciznošću od 1-2 metra. Poboljšane su i notifikacije i poruke, tako da je od Androida 9.0 moguće prikazati sliku u notifikaciji poruke. Još jedna od izmena je poboljšana podrška za više kamera na uređaju, tako da je moguće pristupiti istovremeno strimovima dve ili više fizičkih kamera.

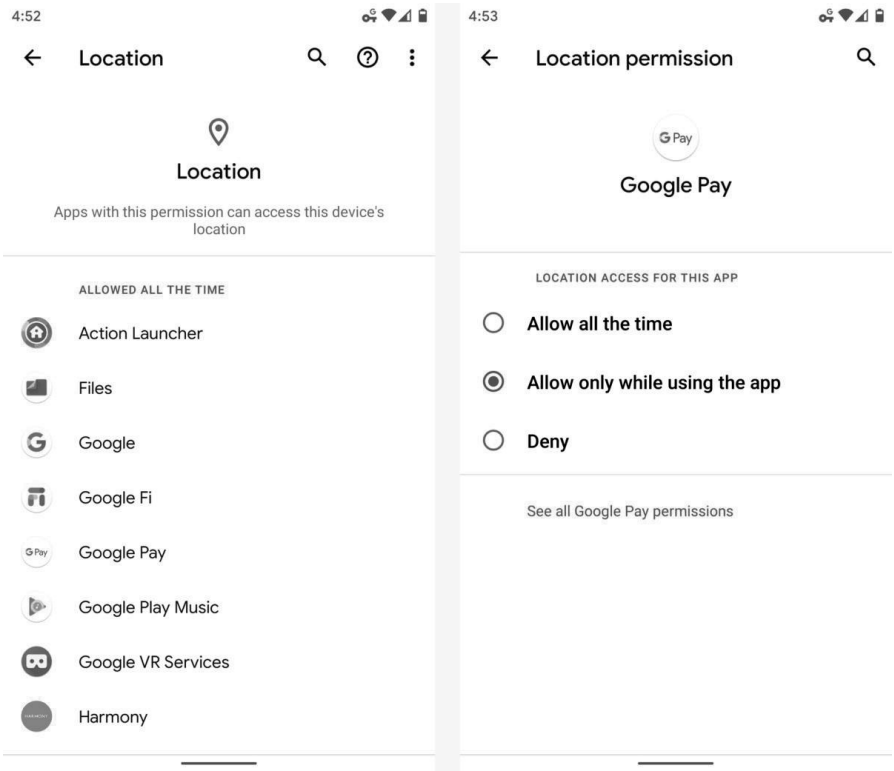
Slika 1.16, Android 9.0 Pie nova opcija pozicioniranja u zatvorenom prostoru (slika preuzeta sa: https://developer.android.com/about/versions/pie/android-9.0)

Osim već navedenih poboljšanja, Pie je uveo veći broj novih funkcionalnosti, među kojima se najviše ističu:

- Poboljšan korisnički interfejs.
- Nove tranzicije prilikom prelaska između različitih aplikacija, ili aktivnosti u okviru jedne aplikacije.
- Procenat baterije uvek prikazan na Always-On ekranu.
- Android Dashboard, koji prikazuje korisniku koliko vremena provodi na uređaju i u određenim aplikacijama, uz opciju da se postave vremenska ograničenja na aplikacije.
- Određen broj eksperimentalnih funkcionalnosti, poput opcije za automatsko uključivanje Bluetooth-a za vreme vožnje.
- Redizajnirani slajder za jačinu zvuka.
- Novi interfejs baziran na većoj upotrebi pokreta prevlačenja (swipe) u odnosu na prethodne verzije.
- Auto-Brightness koji modifikuje osvetljenje ekrana na osnovu navika korisnika.

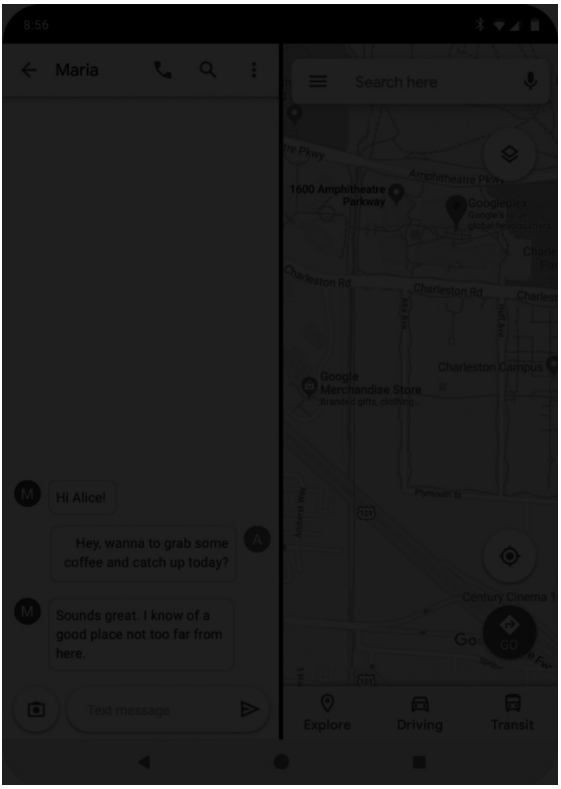
1.1.16. Android 10

Najnovija verzija Androida koja je trenutno aktuelna, dostupna je na tržištu od septembra 2019, godine i poznata je pod jednostavnim nazivom Android 10 (API nivo 29). Jedna od najbitnijih izmena po pitanju sigurnosti se nalazi u kontroli dozvola, gde se korisniku daje puna kontrola da odredi koja aplikacija, kada i kako sme da koristi lokaciju uređaja, kao što je prikazano na slici 1.17.



Slika 1.17, Android 10 kontrola aplikacija koje pristupaju lokaciji
(slika preuzeta sa: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html)

Još jedna od novina je opcija za podršku telefona sa savitljivim ekranima, uz održavanje kontinuiteta prikaza aplikacija u toku savijanja ili ispravljanja ekrana, kao što je prikazano na slici 1.18. Moguće je konfigurisati i emulator virtuelnog uređaja sa savitljivim ekranom.



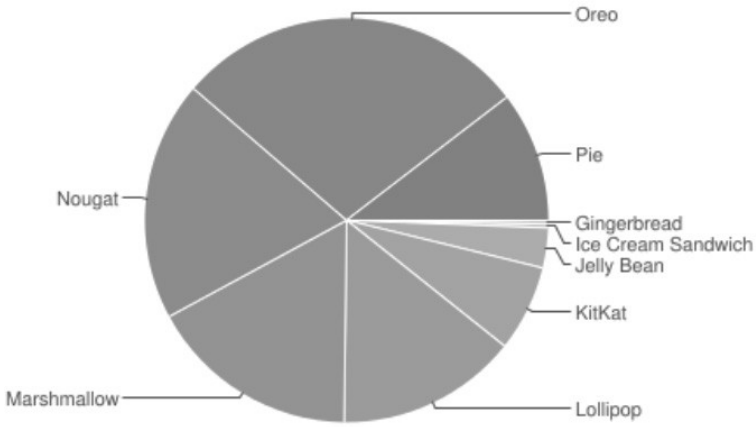
Slika 1.18, Android 10 podrška za savitljive ekrane
(slika preuzeta sa: https://developer.android.com/about/versions/10/highlights)

Još jedna značajna novina je podrška za 5G mobilne mreže, koje su trenutno u procesu uvođenja u većem broju država u svetu. Od drugih novih funkcionalnosti, izdvajaju se:

- Smart Reply opcija, koja se javlja u okviru notifikacije u zavisnosti od konteksta (na primer slanje pametnog odgovora u slučaju notifikacije za poruku, ili otvaranja mape u slučaju da u notifikaciji postoji adresa).
- Tamna tema za uslove lošeg osvetljenja čime se dodatno štedi baterija.
- Deljenje prečica, čime se omogućava direktno deljenje sadržaja sa kontaktom.
- Dinamička dubina za fotografije, čime se može izmeniti zamućenost pozadine nakon snimanja fotografije.
- Poboljšanja u biometrijskog autentifikaciji u aplikacijama.

1.2. Rasprostranjenost Android verzija na tržištu

Veoma važna informacija za programere prilikom razvoja aplikacija za Android je da poznaju situaciju na tržištu, odnosno rasprostranjenost verzija. Ova informacija je uvek dostupna na zvaničnoj stranici za Android programere, koja se nalazi na adresi: https://developer.android.com/about/dashboards/index.html. Na osnovu te informacije programeri mogu da odluče šta je ciljana grupa uređaja i koja je njihova verzija, kako bi maksimalno iskoristili tržište. Na primer, ukoliko je ciljana verzija Android 10, koji ima manje od 1% udela u tržištu, aplikacija možda neće dostići željeni nivo prodaje, jer nije izvesno da će moći da se izvršava na starijim verzijama. Sa druge strane, ukoliko je ciljana verzija na primer Android 4.1 Jelly Bean, biće pokriveno praktično 100% tržišta, odnosno aplikacija će moći da se izvršava na ogromnom broju telefona, ali za takvu aplikaciju nije moguće koristiti naprednije funkcionalnosti koje su uvedene u verzijama nakon Android 4.1. Google periodično radi snimanje trenutne raspodele svih verzija Androida (snimanjem svih aktivnih Android telefona u određenom vremenskom intervalu), a u trenutku pisanja ovog udžbenika dostupne su bile informacije iz početka maja 2019. godine. Grafički prikaz ove statistike prikazan je na slici 1.19, a nakon toga i tabelarno, u tabeli 1-1.



Slika 1.19, raspodela različitih verzija Androida na tržištu
(maj 2019. godine, izvor: https://developer.android.com/about/dashboards)

Potrebno je samo uočiti da se Android 10 ne nalazi prikazan u raspodeli, pošto je izašao na tržište nakon perioda kada je snimano stanje aktivnih Androida. Uopšteno gledano, najnovije verzije Androida često nude odlične i inovativne API-je za naše aplikacije, ali je obično neophodno podržati i starije verzije, bar dok se dovoljno uređaja ne ažurira na novije verzije. U narednim poglavljima će

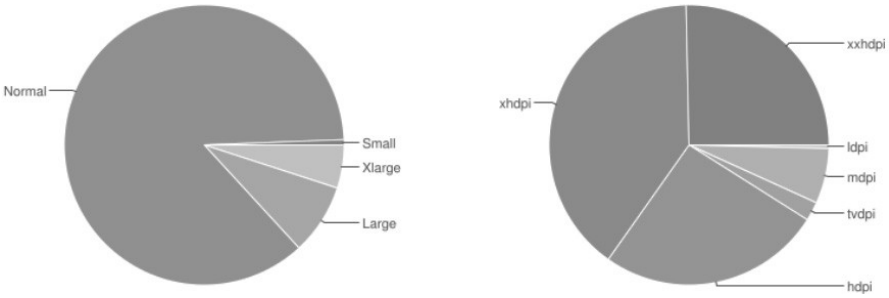
biti objašnjeno i kako se može iskoristiti prednost najnovijih API-ja, uz

kontinuiranu podršku starijih verzija, putem specificiranja minimalne SDK verzije i ciljane SDK verzije unutar AndroidManifest.xml fajla. U praksi se raspodela sa slike 1.19 veoma često koristi za definisanje ove dve vrednosti. Kao primer dobre prakse pokazalo se da bi trebalo podržati 90% aktivnih uređaja.

Verzija	Kodno ime	API	Udeo u tržištu
2.3.3 -	Gingerbread	10	0.30%
2.3.7			
4.0.3 -	Ice Cream Sandwich	15	0.30%
4.0.4			
4.1.x	Jelly Bean	16	1.20%
4.2.x		17	1.50%
4.3		18	0.50%
4.4	KitKat	19	6.90%
5	Lollipop	21	3.00%
5.1		22	11.50%
6	Marshmallow	23	16.90%
7	Nougat	24	11.40%
7.1		25	7.80%
8	Oreo	26	12.90%
8.1		27	15.40%
9	Pie	28	10.40%

Tabela 1-1, udeo različitih verzija Androida u tržištu (maj 2019. godine)

Osim informacije o verziji Androida na uređajima koji se nalaze na tržištu, još jedna bitna stavka je podatak o broju uređaja na tržištu koji imaju određenu konfiguraciju ekrana. Konfiguracija ekrana se definiše preko dva parametra, same veličine ekrana i gustine piksela. Android se izvršava na velikom broju različitih uređaja koji imaju različite veličine ekrana i gustine piksela. Sistem izvršava osnovno skaliranje kako bi adaptirao prikaz korisničkog interfejsa na različitim ekranima, ali kako bi ovaj proces prošao glatko programeri takođe moraju da posvete pažnju tome prilikom razvoja aplikacije. Podatak o konfiguracijama ekrana uređaja koji se nalaze na tržištu je takođe javno dostupan programerima, i može se proveriti na istoj adresi: https://developer.android.com/about/dashboards. Statistika iz maja 2019. godine prikazana je na slici 1.20, a nakon toga i tabelarno, u tabeli 1-2.



Slika 1.20, raspodela veličine ekrana Android uređaja na tržištu (maj 2019. godine, izvor: https://developer.android.com/about/dashboards)

Veličina/gustina	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Ukupno
Mali	0.40%				0.10%	0.10%	0.60%
Normalni		0.90%	0.30%	24.00%	37.70%	23.60%	86.50%
Veliki		2.40%	1.90%	0.60%	1.60%	1.70%	8.20%
Veoma veliki		3.10%		1.30%	0.60%		5.00%
Ukupno	0.40%	6.40%	2.20%	25.90%	40.00%	25.40%	

Tabela 1-2, udeo različitih konfiguracija ekrana Androida u tržištu (maj 2019. godine)

1.3. Udeo Android operativnog sistema na tržištu mobilnih uređaja

Android operativni sistem nije jedini operativni sistem za mobilne uređaje na tržištu. U trenutku izlaska Androida na tržište, konkurencija je bila veoma ozbiljna (slika 1.21). Tada suvereni lider bila je Nokia (Symbian), sa skoro dvotrećinskim udelom u tržištu mobilnih uređaja. Ozbiljnu konkurenciju su dalje činili BlackBerry (RIM), koji je na svom vrhuncu imao i preko 20% udela na tržištu, naročito poslovnih korisnika, kao i Microsoft sa oko 10% i iOS (Apple) sa oko 10%. Operativni sistem Symbian je bio prvi popularni operativni sistem za mobilne telefone na svetu, i smatra se za prvi moderni operativni sistem za mobilne uređaje. Do 2009. godine, udeo Symbian-a je opao na oko 50% tržišta. BlackBerry je takođe bio jedan od pionira u oblasti operativnih sistema za mobilne uređaje, čija su ciljana grupa bili poslovni korisnici. Tada revolucionarna opcija Push email, koja je omogućavala da korisniku mejl bude dostavljen automatski sa servera na telefon, rezultirala je činjenicom da je veliki broj kompanija u svetu svojim zaposlenima davao neki od BlackBerry modela kao poslovni telefon. Iako su imali ogromnu početnu prednost, Symbian i BlackBerry su praktično zbrisani sa tržišta dolaskom novih operativnih sistema. Osim Androida, na tržište je ušao i Apple sa operativnim sistemom iOS i telefonom iPhone 2007. godine.

Slika 1.21, iOS, Android, BlackBerry i Symbian operativni sistemi

Apple je 2007. godine napravio revoluciju na tržištu mobilnih uređaja. Pre iPhone uređaja, pametni telefoni još uvek nisu bili masovni. Danas je iOS drugi najpopularniji operativni sistem za mobilne telefone, sa udelom na tržištu koji u poslednjih 10 godina varira između 14 i 21%. Android je od ulaska na tržište beležio konstantni rast. Oko 7 miliona Android telefona je prodato 2009. godine, da bi ta cifra prešla milijardu prodatih uređaja 2015. godine. Danas je Android ubedljivo prvi operativni sistem za mobilne uređaje u svetu, sa udelom koji poslednjih godina varira između 80 i 85%. Takav trend se očekuje i u narednih nekoliko godina, kao što se može videti iz tabele 1-3 (izvor: https://www.idc.com/promo/smartphone-market-share/os). Svi ostali operativni sistemi zajedno imaju zanemarljiv udeo.

Godina	2017	2018	2019	2020	2021	2022	2023
Android	85.10%	85.10%	87.00%	87.00%	87.20%	87.30%	87.40%
iOS	14.70%	14.90%	13.00%	13.00%	12.80%	12.70%	12.60%
Ostali	0.20%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Ukupno	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

Tabela 1-3, udeo OS na tržištu sa očekivanjima za naredne godine (IDC)

Kada se pogleda udeo tržišta prema proizvođačima mobilnih telefona, prema renomiranoj kompaniji za istraživanje tržišta IDC (International Data Corporation, veb sajt https://www.idc.com/) može se uočiti da dominiraju proizvođači sa dalekog istoka. U tabeli 1-4 prikazan je udeo tržišta po proizvođačima po kvartalima, u periodu od poslednjeg kvartala 2017. do drugog kvartala 2019. godine. Ukupna godišnja prodaja mobilnih telefona iznosi oko 1.5 milijardi isporučenih uređaja. Samsung je već nekoliko godina lider na tržištu, sa otprilike 20% udela na tržištu, odnosno skoro 300 miliona prodatih telefona. Iako je potražnja za Samsung flagship telefonima opala, pošto

većina kupaca traži

jeftinije opcije, srednja klasa predvođena A serijom je doprinela rastu prodaje ovog proizvođača. Iako postoje trgovinske tenzije između Sjedinjenih Američkih Država i Kine, Huawei je održao drugu poziciju sa preko 200 miliona prodatih uređaja. Huawei je vodeći proizvođač na kineskom tržištu, gde drži lidersku poziciju već nekoliko godina. Na trećem mestu proizvođača nalazi se Apple, čija prodaja obično skače u poslednjem kvartalu svake godine, kada izlazi novi model iPhone. Na četvrtom mestu nalazi se Xiaomi, koji najveći deo uređaja isporučuje u Kinu, Indiju i Indoneziju. Još jedan kineski proizvođač, Oppo, koji najveći broj uređaja isporučuje u Kinu i Indiju, nalazi se na petom mestu. Svi ostali proizvođači, među kojima su i giganti poput LG, imaju udeo manji od 6%.

Kvartal	2017Q4	2018Q1	2018Q2	2018Q3	2018Q4	2019Q1	2019Q2
Samsung	18.90%	23.50%	21.00%	20.30%	18.80%	23.00%	22.90%
Huawei	10.70%	11.80%	15.90%	14.60%	16.20%	18.90%	17.70%
Apple	19.60%	15.70%	12.10%	13.20%	18.30%	11.80%	10.20%
Xiaomi	7.10%	8.40%	9.50%	9.50%	6.70%	8.90%	9.70%
Oppo	6.90%	7.40%	8.60%	8.40%	7.90%	7.40%	8.90%
Svi ostali	36.80%	33.20%	32.90%	34.00%	32.00%	30.10%	30.60%
Ukupno	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

Tabela 1-4, udeo proizvođača mobilnih uređaja na tržištu (IDC)

1.4. Tržište mobilnih aplikacija

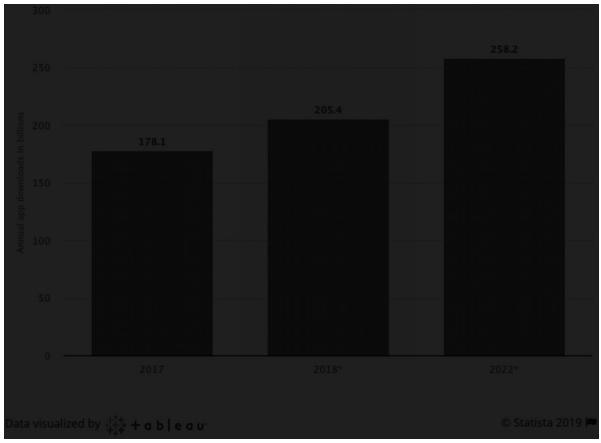
U svetu postoji oko 2.7 milijardi korisnika mobilnih uređaja, pa ne čudi da industrija mobilnih aplikacija cveta. Tržište mobilnih aplikacija već godinama raste stabilnom stopom, bez ikakvih znakova usporavanja u doglednoj budućnosti. Sve je potpuno jasno ukoliko podignemo pogled sa svog telefona i pogledamo oko sebe – većina ljudi takođe je zagledana u svoje mobilne uređaje. Prema nekim studijama, prosečna osoba proverava svoj telefon na svakih 12 minuta, dok svaka deseta osoba proverava telefon na 4 minuta. Mobilni telefoni koriste se svuda – na poslu, u kući, na ulici, dok jedemo, dok se odmaramo u fotelji, u krevetu, čak i u automobilima (što je naročito opasno i izaziva veliki broj saobraćajnih nesreća širom sveta). Od toga, 90% upotrebe mobilnih uređaja se odnosi na upotrebu različitih aplikacija.

Iako tolika vezanost za mobilne uređaje možda nije ohrabrujuća vest za društvo sa psihološkog i sociološkog aspekta, sa stanovišta razvoja mobilnih aplikacija to je izuzetno dobra vest za sve koji žele da se bave ovim poslom. Međutim, kako bismo bili uspešni u ovom poslu, potrebno je da se dese dve ključne stvari: korisnici moraju da preuzmu našu aplikaciju, i zatim da je koriste. Kako bismo

bolje razumeli tržište mobilnih aplikacija, potrebno je poznavati neke statističke podatke koji nam mogu ukazati na veličinu i vrednost samog tržišta:

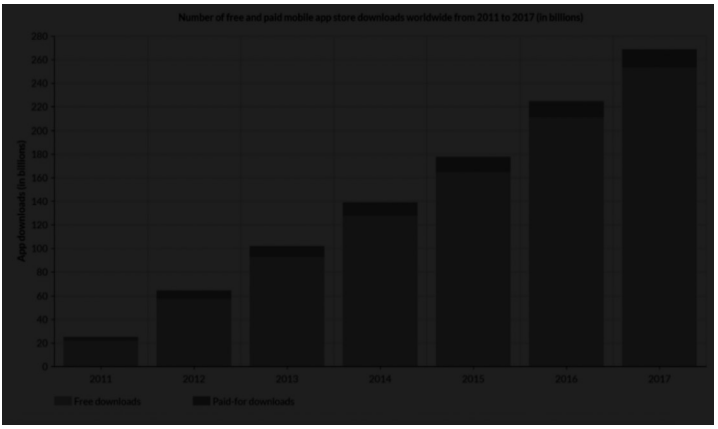
- Očekuje se da mobilne aplikacije donesu prihod od oko 190 milijardi dolara u 2020. godini.
- Google Play Store sadrži oko 2.8 miliona aplikacija za preuzimanje.
- Apple App Store sadrži oko 2.2 miliona aplikacija za preuzimanje.
- Veliki broj korisnika mlađih od 20 godina više od 50 puta dnevno otvori neku od aplikacija na svom uređaju (socijalne mreže, igre, aplikacije za razmenu poruka i slično).
- Polovina svih korisnika mobilnih telefona otvori neku aplikaciju više od 10 puta dnevno.
- Prosečni korisnik mobilnog telefona koristi prosečno 30 aplikacija svakog meseca.
- Prosečni korisnik mobilnog telefona proverava mobilni telefon čak i dok vozi automobil.

Kao izvor za statističke podatke tržišta mobilnih aplikacija korišćen je Statista veb sajt (https://www.statista.com/). Pogled na brojke dovoljno govori o popularnosti mobilnih aplikacija. Ukupan broj preuzimanja mobilnih aplikacija na svim platformama iznosi preko 200 milijardi godišnje, sa tendencijom rasta i u narednim godinama, kao što je prikazano na slici 1.22. Ukoliko se predviđanja eksperata pokažu kao tačna, od 2018. do 2022. godine broj preuzimanja će porasti za oko 25%, sa očekivanim brojem od skoro 260 milijardi preuzimanja aplikacija u 2022. godini. Najveći deo ovih preuzimanja naravno odlazi na Apple i Android, pošto sve ostale platforme imaju zanemarljiv uticaj na priložene brojke.



Slika 1.22, ukupan broj preuzimanja aplikacija na svim platformama (izvor: https://www.statista.com/)

Još jedan bitan parametar jeste odnos preuzimanja besplatnih aplikacija, kao i onih koje se naplaćuju. Cilj razvoja mobilnih aplikacija je svakako zarada novca. Prilikom kreiranja aplikacije, potrebno je osmisliti i strategiju naplate. Jedna od mogućih strategija je naplata preuzimanja aplikacije, i to je verovatno ono što svakom programeru prvo padne na pamet. Međutim, postavlja se pitanje da li će korisnici preuzimati aplikaciju ukoliko se od njih očekuje da plate preuzimanje? Statistički podaci besplatnih i plaćenih preuzimanja aplikacija u posmatranom intervalu od 2011. godine zaključno sa 2018. godinom su prikazani na slici 1.23.



Slika 1.23, ukupan broj besplatnih i plaćenih preuzimanja u intervalu 2011-2017.
(izvor https://www.statista.com/).

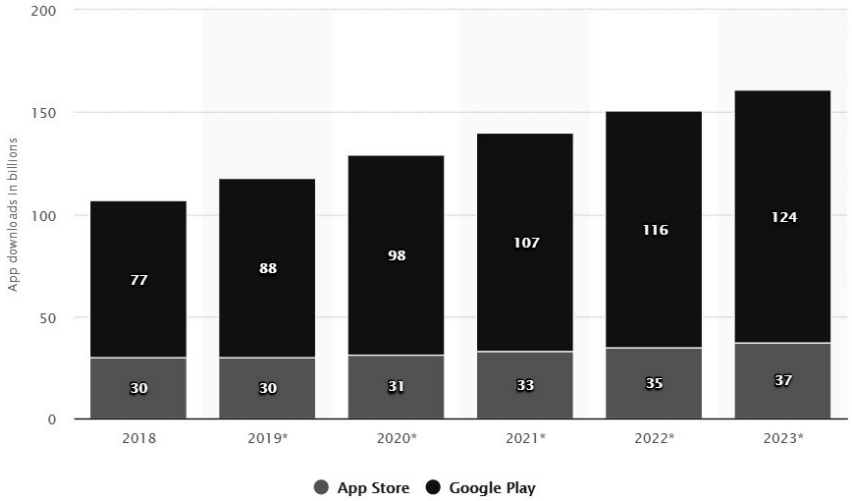
Već na prvi pogled se može uočiti šta su korisnici skloni da preuzimaju, pošto se ogromna većina preuzimanja odnosi na besplatne aplikacije. Prema statistikama, 98% ukupne zarade koje generišu mobilne aplikacije dolazi od besplatnih aplikacija, pošto je jako mali broj korisnika spreman da plati preuzimanje aplikacije. To ne znači da ideja naplate preuzimanja aplikacije u potpunosti loša. U zavisnosti od tipa aplikacije, obe pomenute strategije mogu biti uspešne. Ukoliko se naplaćuje preuzimanje aplikacije, može se očekivati da će manji broj korisnika zaista preuzeti aplikaciju. Razlog tome leži u činjenici da su korisnici navikli da im se aplikacije ponude besplatno – u smislu bez naplate preuzimanja. Dovoljno je da pogledamo naše telefone i razmislimo da li smo nekada instalirali neku aplikaciju čije je preuzimanje bilo naplaćeno. Međutim, korisnici koji su platili preuzimanje aplikacije (iako su u manjini), su mnogo više posvećeni korišćenju aplikacije, pošto niko neće hteti da potroši novac na aplikaciju i da je zatim nikada više ne koristi.

Sa druge strane, nije neophodno naplatiti preuzimanje aplikacije kako bi se zaradio novac. Ukoliko na primer firma koja ima razrađen biznis ponudi

aplikaciju koja će proširiti taj biznis, naročito u smeru poboljšanja zadovoljstva

korisnika, ta aplikacija će najverovatnije biti besplatna, a uvećanje zarade će se odraziti kroz više klijenata primarnog biznisa te firme. Dalje, moguće je zaraditi novac direktno od aplikacije, čak i ako je ponuđena sa besplatnim preuzimanjem. Postoji veći broj strategija, poput kupovine unutar aplikacije (engl. *in-app purchases*) ili preko reklama unutar aplikacije.

Kao što je ranije navedeno, ogromna većina preuzimanja mobilnih aplikacija odlazi na samo dve platforme – Apple i Android. Samim tim, postoje dva mesta odakle korisnici preuzimaju praktično sve aplikacije. Apple korisnici sa iOS uređajima preuzimaju aplikacije sa Apple App Store, dok korisnici Android uređaja svoje aplikacije preuzimaju sa Google Play Store. Na slici 1.24 prikazan je uporedni prikaz broja preuzimanja po platformama za 2018. i 2019. godinu, sa predviđanjima stručnjaka do 2023. godine. Može se uočiti da više preuzimanja dolazi sa Google Play, pošto Android ima više korisnika. Međutim, može se uočiti i da broj preuzimanja raste svake godine na obe platforme, sa tim što Google Play beleži porast od oko 13% u 2019. godini u odnosu na 2018. godinu, dok Apple App Store beleži značajno manji porast, od oko 3% godišnje.

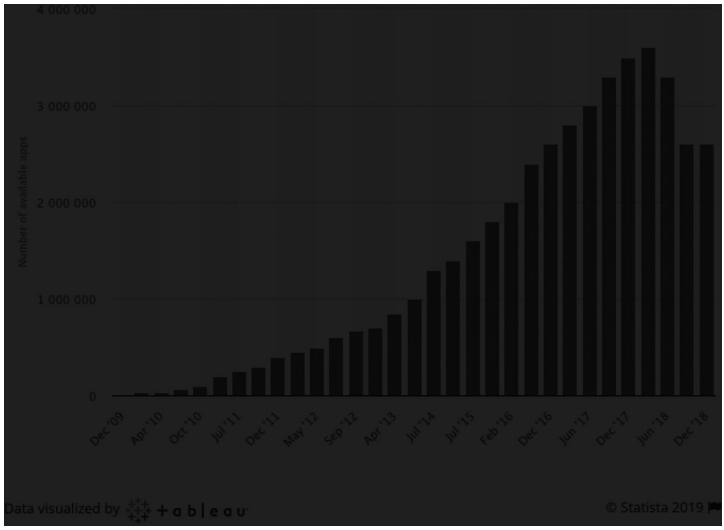


Slika 1.24, uporedni broj preuzimanja sa Google Play i App Store (izvor: https://www.statista.com/)

Ukoliko se posmatraju najčešće preuzimane kategorije aplikacija na obe platforme, najviše preuzimanja imaju igre (oko 25% svih preuzimanja), zatim poslovne aplikacije (oko 10%), zatim edukacione (oko 8.5%), životni stil (oko 8.3%), zabavne aplikacije (oko 6%), dok sve ostale kategorije imaju udeo manji od 5%.

Kako bismo shvatili kolika je zapravo konkurencija, potrebno je pogledati ukupan broj aplikacija dostupnih za preuzimanje. Statistički podaci o broju

aplikacija koje su dostupne za preuzimanje na Google Play u periodu od 2009. godine do kraja 2018. godine su prikazani na slici 1.25. Sa grafika se može uočiti ogroman rast broja aplikacija, od skromnih početaka, preko prvih milion aplikacija (2014. godine), zatim cifre od dva miliona dostupnih aplikacija koja je dostignuta 2016. godine, sve do maksimuma od oko 3.5 miliona dostupnih aplikacija 2017. godine. Međutim, od druge polovine 2017. godine počinje da se beleži pad broja dostupnih aplikacija. Iako to možda deluje neočekivano, rezultati potvrđuju ogromnu konkurenciju aplikacija na Google Play – neke aplikacije prosto nisu bile sposobne da prežive. Nasuprot tome, kao što je ranije navedeno, ukupan broj preuzimanja aplikacija raste svake godine. Trenutno je raspoloživo oko 2.5 miliona aplikacija na Google Play, i ne očekuje se dalji pad. Pretpostavlja se da će broj aplikacija ostati na ovom nivou, a zatim u narednim godinama početi ponovo da raste.



Slika 1.25, broj aplikacija dostupnih na Google Play Store
(izvor: https://www.statista.com/)

2. Android platforma

Programer bi trebalo dobro da upozna platformu sa kojom radi, jer isključivo sa dobrim razumevanjem sistema može izvući maksimum iz svojih aplikacija. Programiranje za Android se najčešće radi u programskom jeziku Java, međutim, postoje određene specifičnosti sa kojima se čisti Java programeri nisu prethodno susretali, i koje je neophodno uvesti i objasniti na početku. U ovom poglavlju opisana je arhitektura Android platforme, softverski stek, zatim izvršno okruženje (engl. *runtime*), kao i osnovni detalji Android aplikacija.

2.1. Arhitektura Android platforme

Android platforma je zasnovana na softverskom steku na čijem se dnu nalazi Linux jezgro. Na slici 2.1 prikazani su osnovni slojevi i komponente platforme.

Slika 2.1, Android softverski stek
(izvor: https://developer.android.com/guide/platform)

Osnova Android platforme leži u Linux jezgru. Ovaj modul zadužen je za upravljanje ključnim sistemskim servisima i resursima, poput memorije, procesa, niti, hardvera i potrošnje energije. ART (engl. *Android Runtime*) se u potpunosti oslanja na Linux kernel za ove ključne sistemske servise. Upotreba Linux jezgra omogućava Androidu da iskoristi njegove ključne sigurnosne funkcionalnosti i omogućava proizvođačima telefona da razvijaju hardverske drajvere za poznato jezgro.

Linux jezgro se koristi godinama u velikom broju različitih okruženja koja su osetljiva po pitanju sigurnosti, i danas se smatra da je to stabilno i sigurno jezgro kome veruje veliki broj korporacija. Sa stanovišta mobilnih uređaja, Linux jezgro pruža nekoliko ključnih sigurnosnih funkcija, od kojih su najvažniji model dozvola (privilegija) na nivou korisnika, izolacija procesa, kao i sprečavanje da jedan korisnik čita fajlove drugog korisnika, pošto je Linux višekorisnički operativni sistem.

Iznad Linux jezgra nalazi se sloj apstrakcije hardvera (engl. HAL – *hardware abstraction layer*). Ovaj sloj je zadužen da obezbedi set standardnih interfejsa koji otkrivaju hardverske karakteristike Java API okviru višeg nivoa. U ovom sloju se nalazi više biblioteka, a svaka biblioteka je zadužena da implementira interfejs ka određenoj hardverskoj komponenti, poput modula za kameru, Bluetooth ili nekog od senzora dostupnih na uređaju. Kada Java API napravi poziv da pristupi određenoj hardverskoj komponenti uređaja, Android sistem učitava biblioteku za traženu komponentu.

Iznad sloja apstrakcije hardvera nalazi se Android izvršno okruženje. Počev od Android verzije 5.0 (API nivo 21), svaka aplikacija se izvršava u svom procesu i sa svojom instancom ART. ART je implementiran na takav način da može da pokrene više virtuelnih mašina na uređajima sa ograničenom memorijom, izvršavanjem DEX fajlova. DEX je zapravo bajtkod dizajniran specijalno za Android, tako što je optimizovan za minimalni memorijski otisak. Pre Android 5.0, kao izvršno okruženje koristio se Dalvik. Ukoliko se aplikacija ispravno izvršava na ART, onda bi trebalo da radi bez problema i na Dalviku, ali obrnuto ne mora da važi. Android sadrži i skup ključnih izvršnih biblioteka, koje pružaju veći deo funkcionalnosti programskog jezika Java, koji je najbliži Java verziji 8.

Na ovom nivou se, pored ART, nalazi i skup izvornih C/C++ biblioteka. Mnoge sistemske komponente i servisi Androida su napisani u izvornom kodu koji zahteva ove izvorne biblioteke. Android platforma kroz Java API otkriva aplikacijama funkcionalnost nekih od ovih biblioteka. Na primer, moguće je pristupiti OpenGL ES modulu (Open Graphics Library – podrška za 2D i 3D grafiku visokih performansi) kroz Java OpenGL API kako bi se u aplikaciji dodala podrška za iscrtavanje i manipulisanje 2D i 3D grafikom. Moguće je i razvijati aplikacije direktno upotrebom C ili C++ koda, u tom slučaju se može koristiti Android NDK za pristup izvornim bibliotekama.

Sledeći sloj predstavlja Java API okvir, preko koga je programerima dostupan ceo skup funkcionalnosti Androida. Preko ovog Java okvira se na jednostavan način mogu izgraditi aplikacije, putem upotrebe modularnih sistemskih komponenta i servisa, od kojih su najbitniji:

- View System, koji je bogat i lako proširiv. Pomoću njega se jednostavno pravi korisnički interfejs aplikacije, upotrebom lista, rasporeda komponenti, tekstualnih polja, dugmića itd.
- Resource Manager, koji omogućava pristup resursima aplikacije koji nisu kod, poput lokalizovanih stringova, slika i razmeštaja komponenti.
- Notification Manager, koji omogućava svim aplikacijama da prikažu obaveštenja u statusnoj liniji.
- Activity Manager, koji je zadužen za menadžment životnog ciklusa aplikacije i obezbeđuje zajednički navigacioni stek (engl. *back stack*).
- Content Provider komponente, koje omogućavaju aplikacijama da pristupe podacima drugih aplikacija, odnosno omogućavaju deljenje podataka.

Potrebno je napomenuti da programeri prilikom razvoja svojih aplikacija koriste identični API okvir koji koriste i sistemske aplikacije Androida.

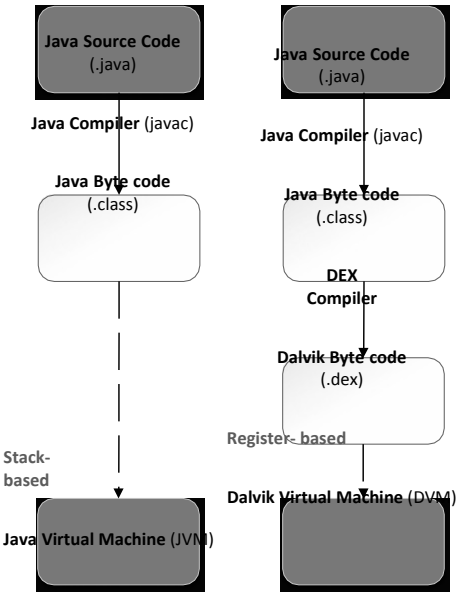
Na vrhu steka se nalazi nivo sistemskih aplikacija. To je skup predefinisanih osnovnih aplikacija koje dolaze zajedno sa Androidom, poput mejl klijenta, SMS poruka, kalendara, kamere, Internet pretraživača, kontakata, mapa i slično. Ove preinstalirane aplikacije nemaju poseban status u odnosu na aplikacije koje korisnik kasnije odluči da instalira, tako da je moguće da se promeni podrazumevani veb čitač ili podrazumevana tastatura nekom aplikacijom razvijenom od treće strane, uz određene izuzetke poput sistemske aplikacije za podešavanja – Settings. Sistemske aplikacije, osim što služe kao korisničke aplikacije, pružaju ključne funkcionalnosti koje programeri mogu da iskoriste i pristupe im iz aplikacije koju razvijaju. Na primer, ukoliko je potrebno iz aplikacije poslati SMS poruku ili napraviti fotografiju kamerom, nije potrebno samostalno razviti tu funkcionalnost, već se može pozvati postojeća SMS aplikacija da prosledi poruku željenoj osobi, odnosno može se pozvati postojeća Android aplikacija kamera pomoću koje ćemo napraviti fotografiju.

2.2. Android Runtime

Izvršno okruženje koje koriste aplikacije i neki sistemski resursi na Androidu se naziva Android runtime (ART). Iako je ART već pomenut u prethodnom poglavlju, zaslužuje da mu se posveti malo više pažnje. Kao što je već navedeno, ART je dostupan počev od verzije Android 5.0. I ART i njegov prethodnik Dalvik

su kreirani specijalno u okviru projekta Android, prema specifičnim zahtevima mobilne platforme. ART izvršava DEX bajtkod (DEX – Dalvik Executable format). Sa stanovišta kompatibilnosti, i ART i Dalvik mogu da izvršavaju DEX bajtkod, tako da aplikacije koje su razvijene za Dalvik rade i kada se izvršavaju na ART, mada obrnuto ne važi.

Dalvik virtuelna mašina je veoma slična Java virtuelnoj mašini (JVM). U slučaju standardne Jave, izvorni Java kod se prevodi u Java bajtkod, koga zatim interpretira JVM. Po istoj logici, Dalvik virtuelna mašina izvršava DEX bajtkod. Razlika između običnog i DEX bajtkoda leži u činjenici da je DEX optimizovan za mobilne uređaje (mali memorijski otisak, štednja energije, performanse). Uporedni prikaz standardne JVM i Dalvik virtuelne mašine, sa procesom prevođenja, prikazan je na slici 2.2. Androidova virtuelna mašina u potpunosti zavisi od Linux jezgra po pitanju upravljanja nitima i menadžmenta memorije. Svaka aplikacija se, uz pomoć Dalvik ili ART virtuelne mašine, izvršava u zasebnom procesu u okviru izvršnog okruženja.



Slika 2.2, uporedni prikaz JVM i Dalvik VM

ART je dodatno unapredio Dalvik virtuelnu mašinu. Uvođenjem AOT (Ahead-of-Time) prevođenja, poboljšane su performanse aplikacija, i smanjeno je vreme instalacije aplikacija. Za vreme instalacije, ART prevodi aplikacije upotrebom dex2oat alata. Ovaj alat prihvata .dex fajlove kao ulaz i generiše prevedeni izvršni fajl (.oat) za ciljani uređaj. Još jedna stvar koja je značajno poboljšana sa ART je sakupljanje đubreta (engl. *Garbage Collection* - GC). GC može drastično da degradira performanse aplikacije na mobilnim uređajima, što za

rezultat ima loš

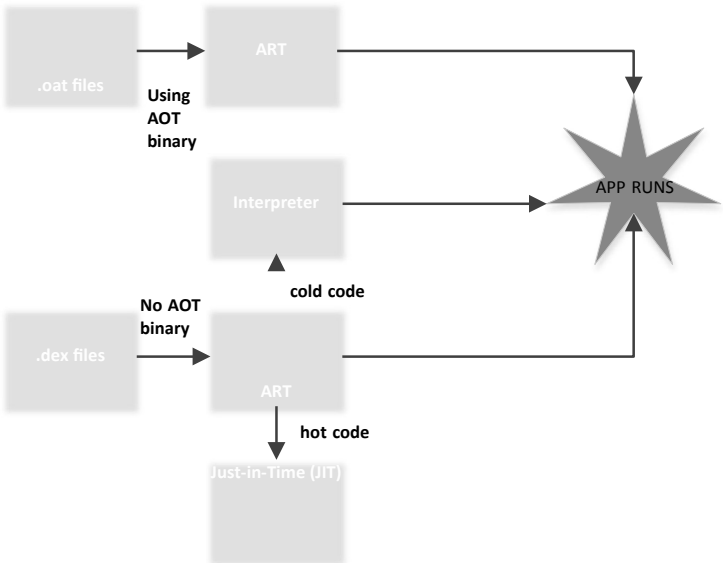
odziv korisničkog interfejsa i „seckanje“ prikaza. Poboljšanje je postignuto optimizacijama poput paralelnog procesiranja i kompaktnosti kako bi se smanjila potrošnja memorije. Na kraju, ART nudi veći broj funkcionalnosti koje olakšavaju razvoj i debagovanje aplikacije.

ART u sebi sadrži JIT (Just-in-Time) kompajler, koji je komplementaran AOT kompajleru. JIT poboljšava performanse samog izvršnog okruženja, čuva memoriju i ubrzava ažuriranje aplikacije. JIT dodatno poboljšava AOT kompajler tako što se izbegava usporavanje sistema za vreme automatskog ažuriranja aplikacija. Prikaz arhitekture JIT prikazan je na slici 2.3.

Pojednostavljeno gledano, JIT u praksi funkcioniše na sledeći način. Kada korisnik pokrene aplikaciju, okida se ART koji učitava .dex fajl. U zavisnosti da li je .oat fajl (AOT verzija ,dex fajla) raspoloživ ili ne moguća su dva scenarija:

- Ukoliko je .oat fajl raspoloživ, ART ga koristi direktno. Iako se .oat fajlovi generišu redovno, to ne znači da uvek sadrže prevedeni kod.
- Ukoliko .oat fajl ne sadrži prevedeni kod, ART koristi JIT i interpreter kako bi izvršio .dex fajl.

JIT podaci se čuvaju u fajlu u sistemskom direktorijumu kome samo aplikacija može da pristupi. AOT kompajler (dex2oat) parsira taj fajl i upravlja njegovom kompilacijom. Drugim rečima, kod uređaja koji koriste ART, veliki deo izvršnog bajtkoda se prevodi već u vreme instalacije. Prilikom narednog pokretanja aplikacije nije potrebno kompletno prevesti .dex izvršni bajtkod aplikacije, čime se značajno ubrzava rad.



Slika 2.3, JIT arhitektura

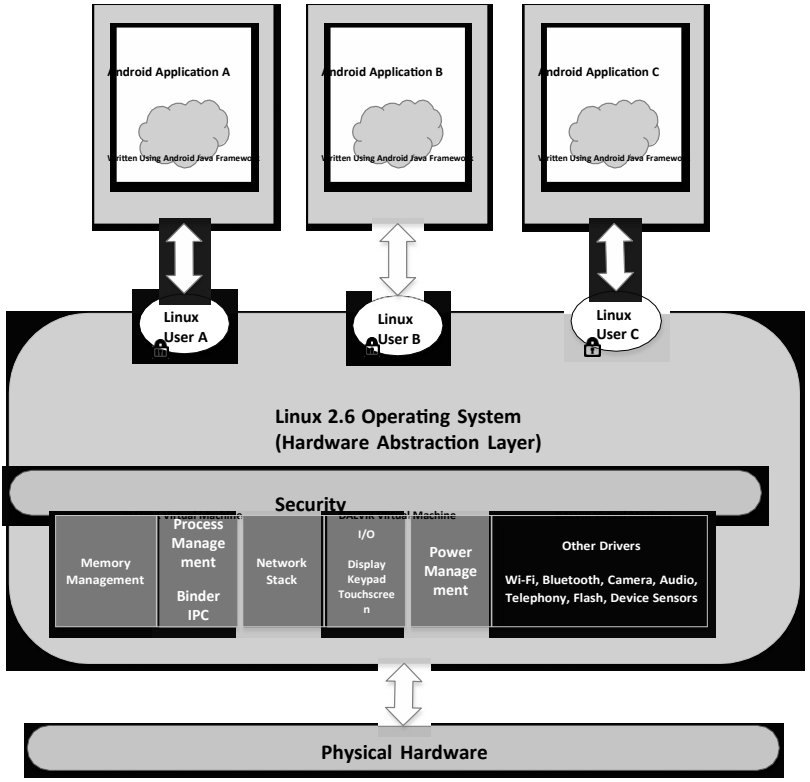
2.3. Android aplikacije

Android aplikacije se mogu pisati u programskim jezicima Java, Kotlin ili C++. Kotlin svakim danom dobija na popularnosti, međutim, još uvek je Java najčešće korišćeni jezik, pa je u ovoj knjizi fokus na razvoju aplikacija upotrebom Jave. Za razliku od standardne Jave SE, Android ne podržava neke biblioteke (na primer biblioteke za razvoj korisničkog interfejsa, AWT i Swing, za koje Android ima svoju posebnu biblioteku). Android SDK prevodi kod aplikacije, zajedno sa podacima i resursima u APK (Android Package), koji predstavlja arhivu sa .apk sufiksom. U APK fajlu se nalazi kompletan sadržaj koji je potreban uređaju da instalira tu aplikaciju.

Svaka Android aplikacija se izvršava u svom Linux procesu (slika 2.4), zaštićena Androidovim sigurnosnim funkcijama:

- Android OS je višekorisnički Linux sistem, u kome se svaka aplikacija posmatra kao drugi korisnik.
- Sistem svakoj aplikaciji dodeljuje jedinstveni korisnički Linux ID (ovaj ID koristi isključivo sistem i nije poznat samoj aplikaciji). Sistem postavlja privilegije pristupa svim fajlovima u aplikaciji, tako da samo korisnički ID dodeljen toj aplikaciji može da im pristupi. Drugim rečima, fajlovi jedne aplikacije su dostupni samo toj aplikaciji.
- Svaki proces poseduje svoju zasebnu virtuelnu mašinu, pa se aplikacija izvršava izolovano od drugih aplikacija.
- Android pokreće proces kada je potrebno izvršiti bilo koju komponentu aplikacije, a završava ga kada više nije potreban ili kada sistemu nedostaje memorije za nesmetan rad drugih aplikacija.

Android koristi princip najmanje privilegije. To znači da aplikacija podrazumevano ima pristup samo komponentama koje su joj potrebne za rad i ničemu više. Na taj način se kreira veoma sigurno okruženje u kome aplikacija ne može pristupiti delovima sistema za koje joj nije data dozvola. Ukoliko je aplikaciji potreban pristup nekom delu Android sistema, poput kamere, lokacije ili konekcije na Internet, ona može zahtevati dozvolu od korisnika, koji mora eksplicitno potvrditi (ili opovrgnuti) te dozvole.



Slika 2.4, izvršavanje Android aplikacija u zasebnim Linux procesima

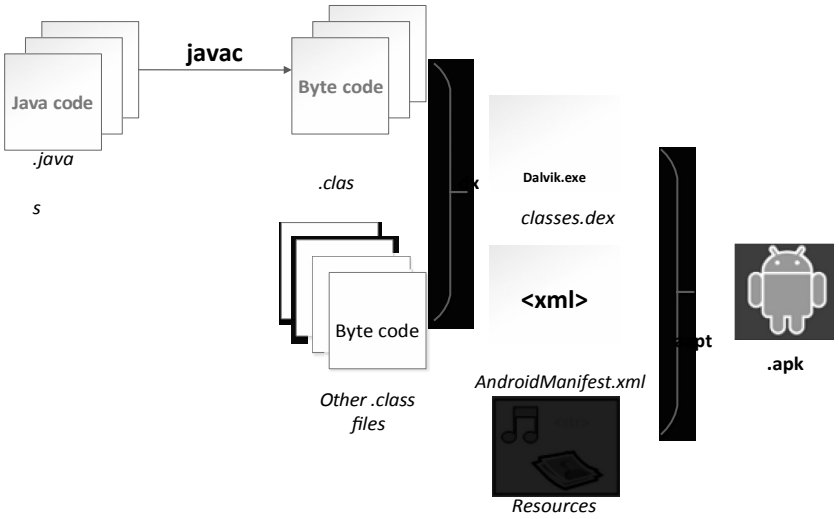
Svaka aplikacija se sastoji od komponenti. Postoje četiri osnovne komponente: aktivnosti, servisi, prijemnici obaveštenja (engl. *Broadcast receivers*) i komponente koje pružaju sadržaj (engl. *Content providers*). Svaki tip komponente ima svoju svrhu i svoj odgovarajući životni ciklus, koji definiše kako se komponenta kreira odnosno uništava. Osim Java fajlova koji sadrže kod, aplikacija sadrži i resurse poput dodatnih fajlova ili statičkog sadržaja koji kod aplikacije koristi. Ovi resursi mogu biti slike, definicije rasporeda komponenti (engl. *layout*), lokalizovani stringovi i slično. Android razdvaja resurse od koda, kako bi se mogli održavati nezavisno. Vrlo često se koriste i alternativni resursi za različite konfiguracije uređaja, koji se grupišu u odgovarajuće direktorijume. Android onda u vreme izvršavanja koristi odgovarajuće resurse na osnovu stvarne konfiguracije uređaja na kome se aplikacija trenutno izvršava. Svakom resursu se iz koda može pristupiti preko ID koji se generiše i čuva u R.java klasi.

Dodatno, svaka aplikacija mora da sadrži i posebni XML fajl pod strogo definisanim nazivom AndroidManifest.xml. U ovom fajlu su definisane osnovne informacije o aplikaciji koje su potrebne Android operativnom sistemu

za ispravno izvršavanje aplikacije, poput imena aplikacije, definicije komponenti,

dozvola koje su potrebne aplikaciji za pristup zaštićenim delovima Android sistema i slično.

Android SDK prevodi sve ove komponente u APK, kao što je prikazano na slici 2.5. APK je format fajla koji Android koristi za distribuciju i instaliranje aplikacija, i sadrži sve što je potrebno kako bi Android ispravno instalirao aplikaciju na uređaju.



Slika 2.5, struktura APK

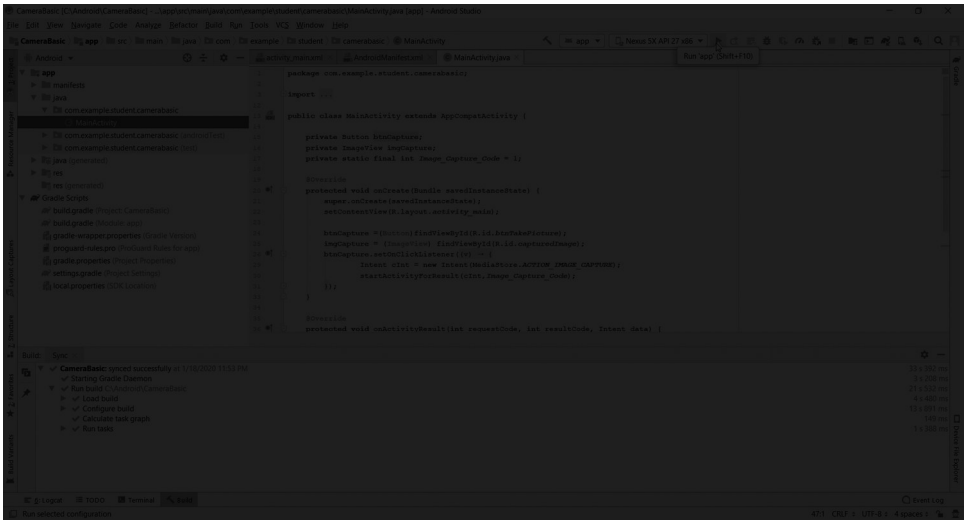
2.4. Okruženje za razvoj

Za razvoj Android aplikacija potrebno je odgovarajuće okruženje za razvoj (engl. IDE – *Integrated Development Environment*). Google je razvio veoma korisno okruženje za Android programere, pod nazivom Android Studio. Android Studio je zvanični IDE za razvoj Android aplikacija i jedini kojem je moguće razviti aplikacije za sve verzije Android operativnog sistema. Ranije je bilo moguće koristiti i Eclipse IDE za razvoj Android aplikacija upotrebom ADT (Android Developer Tools) proširenja, međutim Google je zvanično uskratio podršku ovom okruženju 2015. godine. To se poklapa sa izlaskom prve zvanične verzije Android Studio 1.0, koja je izašla na tržište u decembru 2014. Pošto Google nije nastavio dalje da razvija ADT, Eclipse IDE je i dalje moguće ograničeno koristiti i razvijati aplikacije zaključno sa verzijom Android 4.0, mada to nije preporučljivo.

Android Studio je okruženje bazirano na IntelliJ IDEA, a omogućava programiranje u programskim jezicima Java i Kotlin, uz podršku i za

programski

jezik C++. Izgled okruženja prikazan je na slici 2.6. Okruženje je dizajnirano i organizovano na takav način da su najčešće upotrebljavane funkcije vidljive i lako dostupne, mada i programer sam može organizovati okruženje na način kako mu najviše odgovara.



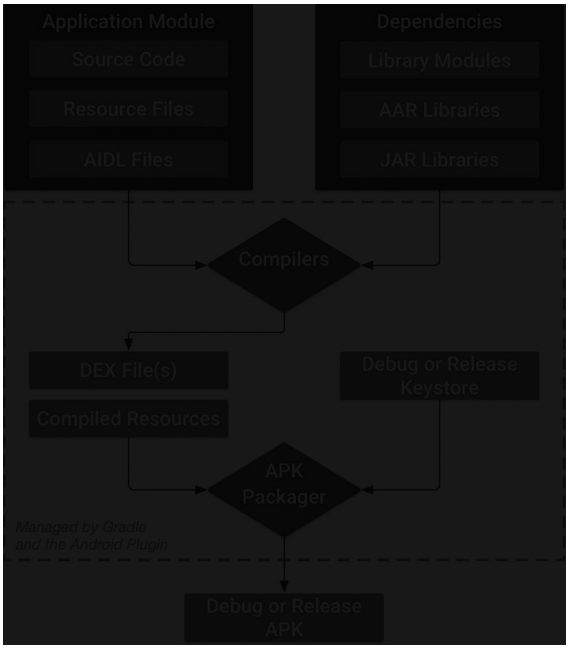
Slika 2.6, Android Studio IDE

Android studio kompajlira izvorni kod zajedno sa resursima aplikacije, pakuje ih u APK koji se može testirati, potpisati i distribuirati. Proces je prikazan na slici
2.7. Android Studio koristi Gradle za automatizaciju i vođenje ovog procesa. Sam proces pravljenja build-a aplikacije uključuje veći broj alata koji prevode projekat u APK. Tipičan proces uključuje sledeće opšte korake:

- Kompajleri su zaduženi da izvorni kod prevedu u DEX fajlove koji sadrže bajtkod koji se izvršava na Android uređaju, a sve ostalo u kompajlirane resurse.
- APK Packager komponenta spaja DEX fajlove i kompajlirane resurse u jedinstveni APK. Pre nego što se aplikacija instalira ili distribuira na Android uređaje, neophodno je da bude potpisana (engl. *signed*).
- APK Packager potpisuje APK upotrebom ili debug ili release ključa
    - Ukoliko je verzija aplikacije potrebna samo za testiranje, APK Packager potpisuje aplikaciju sa debug ključem (Android Studio automatski konfiguriše nove projekte sa debug ključem).
    - Ukoliko je verzija aplikacije namenjena eksternoj distribuciju (engl. *release version*), APK Packager potpisuje aplikaciju sa release ključem (koji se mora kreirati u okviru Android Studio okruženja).

- Pre generisanja finalne verzije APK, Packager dodatno optimizuje aplikaciju da koristi manje memorije prilikom izvršavanja na uređaju.

Na kraju ovog procesa, kao rezultat se dobija ili debug APK ili release APK koji se mogu koristiti za instalaciju, testiranje i distribuciju drugim korisnicima.



Slika 2.7, proces kreiranja aplikacije u Android Studio IDE
(izvor: https://developer.android.com/studio/build)

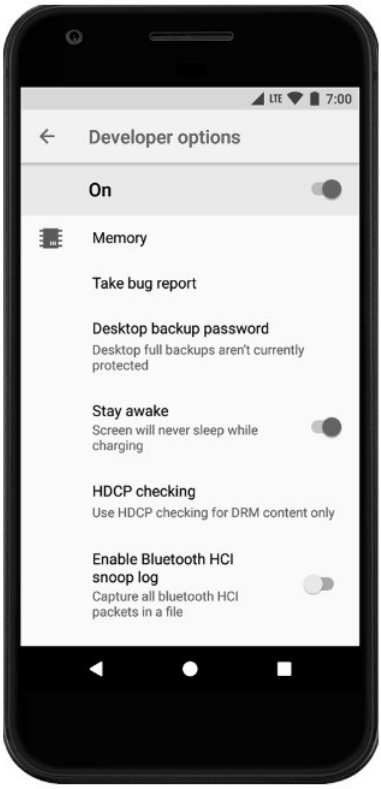
2.5. Android uređaji i emulatori

Razvoj mobilnih aplikacija za Android ima još jednu razliku u poređenju sa tradicionalnim Java aplikacijama. Kako bi se razvijena aplikacija mogla pokrenuti i testirati, potrebno je imati stvarni uređaj na kome se aplikacija biti instalirana. To je relativno jednostavno uraditi tako što se Android uređaj preko USB kabla poveže sa računarom, a zatim se Android Studio preko ADB (Android Debug Bridge) poveže sa uređajem. ADB je alat koji omogućava komunikaciju sa uređajem i omogućava veliki broj akcija sa tim uređajem, poput instalacije i debagovanja aplikacija. Realizovan je kao klijent-server program, i sastoji se od sledećih komponenti:

- Klijent – izvršava se na računaru na kome se razvija aplikacija i šalje komande.

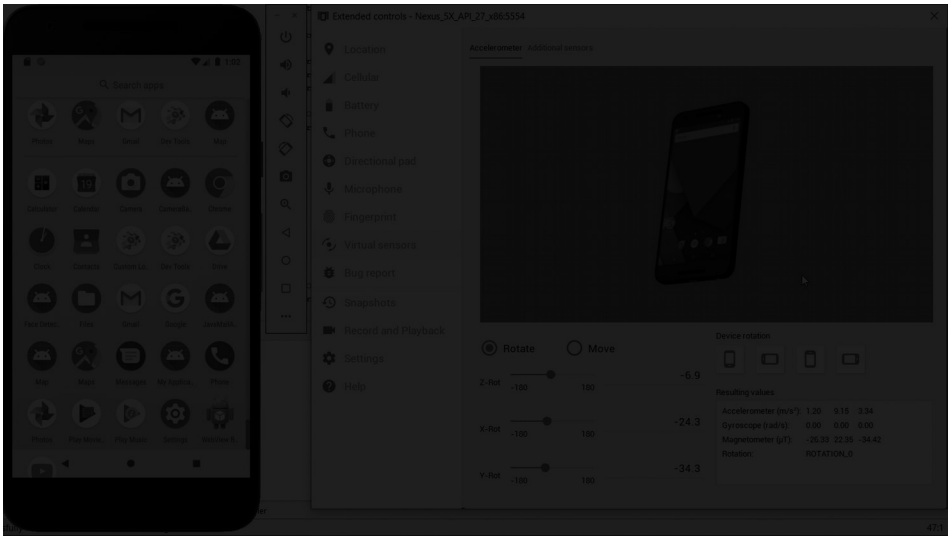
- Daemon (abdb) – izvršava se kao pozadinski proces na uređaju i izvršava komande na njemu.
- Server – služi za vođenje komunikacije između klijenta i daemon-a, izvršava se kao pozadinski proces na računaru na kome se razvija aplikacija.

Kako bi se na uređaju koji je povezan preko USB koristio ADB, potrebno je uključiti opciju USB debugging u okviru podešavanja uređaja, u skrivenoj sekciji Developer options (slika 2.8). Kako bi se ova sekcija aktivirala i postala vidljiva, na najvećem broju uređaja potrebno je otići na Settings – About phone, pronaći u okviru ove sekcije Build number i dodirnuti ga 7 puta. Prilikom sledećeg povratka u Settings sekciju, opcija Developer options će biti dostupna na dnu. Na nekim uređajima se ova opcija može nalaziti na drugom mestu i sa nešto drugačijim imenom. Sada se prilikom povezivanja uređaja preko USB mogu izvršavati i ADB komande. Veliki broj Android programera se prilikom razvoja aplikacija odlučuje upravo za ovu varijantu, da debaguje aplikaciju direktno na fizičkom uređaju.



Slika 2.8, skrivena sekcija Developer options
(izvor: https://developer.android.com/studio/debug/dev-options)

Međutim, fizički uređaj ponekada nije adekvatna opcija. Na primer, ukoliko ciljani uređaj nije dostupan iz bilo kog razloga (imamo uređaj sa Android 8.0, a ciljana platforma je Android 9.0), ili ukoliko želimo da proverimo aplikaciju na više različitih uređaja sa različitim API nivoima, pa nije praktično imati veliki broj fizičkih uređaja. U tom slučaju, moguće je koristiti Android Emulator, koji simulira Android uređaje na računaru. Emulator pruža skoro sve funkcionalnosti stvarnog Android uređaja. Moguće je simulirati pozive, tekstualne poruke, specificirati lokaciju uređaja (moguće je učitati rutu i simulirati kretanje), različite tipove mreže i jačinu signala, simulirati rotaciju i fizičke pokrete kako bi se testirali senzori i slično (slika 2.9). Debagovanje i testiranje aplikacije na emulatoru je iz više razloga brže i lakše neko u slučaju fizičkog uređaja – na primer, aplikacija se može instalirati i podaci se mogu prebaciti na emulator brže nego na uređaj povezan preko USB.



Slika 2.9, Android Emulator sa dodatnim funkcionalnostima

Emulator se isporučuje sa već predefinisanim konfiguracijama za različite Android telefone, tablete, satove i TV uređaje. Što se tiče telefona, ponuđeni su predefinisani Google telefoni tipa Nexus. Moguće je simulirati i uređaje drugih proizvođača podešavanjem odgovarajućih fizičkih karakteristika i upotrebom skinova. Potrebno je na kraju napomenuti da Android Emulator ima daleko veće hardverske zahteve od samog Android Studio okruženja. Konkretno, zahteva se 64-bitni procesor, minimum 8GB RAM (4GB RAM je u teoriji dovoljno da se pokrene Android Studio i Emulator, ali u praksi je neprihvatljivo sporo). Za hardversku akceleraciju dodatno je potrebno:

- Za Intel procesore: Intel procesor sa podrškom za Intel VT-x, Intel EM64T i Execute Disabe (XD) Bit funkcionalnostima.

- Za AMD procesore: AMD procesor sa podrškom za AMD Virtuelizaciju (AMD-V).

Od juna 2019. godine počela je da se ukida podrška za 32-bitne Windows operativne sisteme. Osnovna podrška će trajati do kraja 2020. godine, uz ispravke kritičnih bagova, ali se nove funkcionalnosti neće dodavati. Zbog toga, preporuka je da se razvoj Android aplikacija i upotreba Android Emulatora vrši na uređaju sa 64-bitnim Windows okruženjem.

3. Android Studio

Android Studio je zvanični IDE za razvoj Android aplikacija, kao što je već pomenuto u prethodnom poglavlju. Okruženje se može preuzeti sa lokacije: https://developer.android.com/studio/. U trenutku pisanja ovog udžbenika, aktuelna verzija okruženja bila je 3.5.2, iz oktobra 2019. godine. Svi primeri koda dati u nastavku ovog udžbenika implementirani su u ovoj verziji okruženja (slika 3.1).

Slika 3.1, verzija Android Studio okruženja korišćena u ovoj knjizi

U nastavku ovog poglavlja dato je detaljno uputstvo kako se instalira i konfiguriše okruženje, objašnjene su glavne komponente okruženja, opisano kako se konfiguriše emulator i dat je prvi jednostavan primer funkcionalne Android aplikacije.

3.1. Instalacija okruženja

Pre preuzimanja instalacionog fajla, potrebno je proveriti da li računar na kome se Android Studio instalira ispunjava hardverske zahteve. Za računar sa Windows operativnim sistemom zahtevi su sledeći:

- Operativni sistem Microsoft Windows 7/8/10 (preporučljivo 64-bitni).
- 4 GB RAM minimum, preporučljivo 8 GB RAM ili više (zbog emulatora).
- 2 GB dostupnog prostora na disku minimum (IDE + Android SDK).
- Minimalna rezolucija ekrana 1280 x 800.

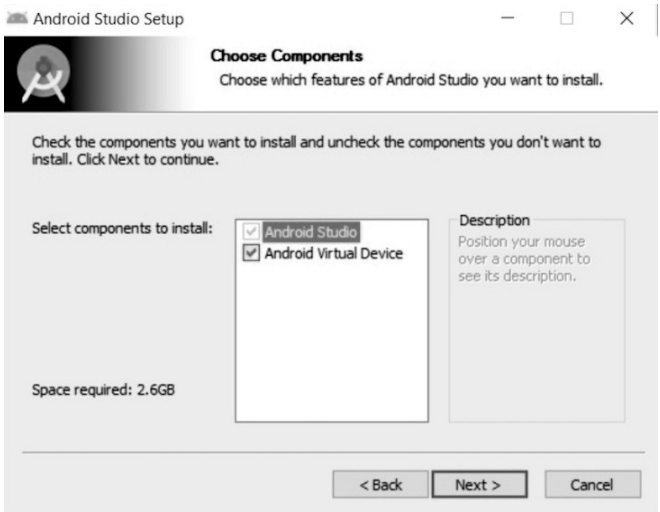
Prilikom instalacije okruženja na Windows računaru, nakon preuzimanja instalacionog fajla najnovije dostupne stabilne verzije sa adrese date na početku ovog poglavlja, potrebno je izvršiti dupli klik na .exe fajl. Sledeći koraci ilustruju proces instalacije koji je identičan za sve 3.x verzije Android Studio

okruženja.

Instalacija započinje prikazom dijaloga dobrodošlice, gde se korisniku savetuje da isključi sve druge aktivne aplikacije za vreme procesa instalacije, kao što je prikazano na slici 3.2. Klikom na Next dugme pokreće se čarobnjak koji će voditi proces instalacije.

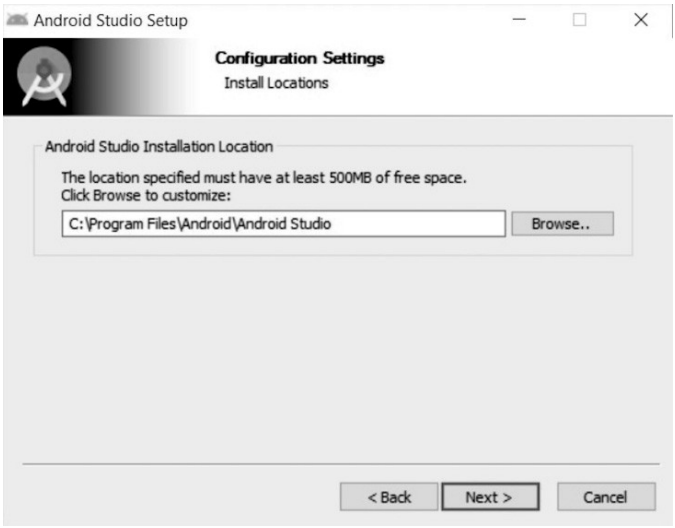
Slika 3.2, početni ekran procesa instalacije

Sledeći ekran nudi opciju za uključivanje/isključivanje komponenti koje će biti instalirane. Android Studio komponenta je obavezna, dok se opciono može isključiti Android Virtual Device. Preporučeno je ipak ostaviti podrazumevana podešavanja instalacije, kao što je prikazano na slici 3.3.

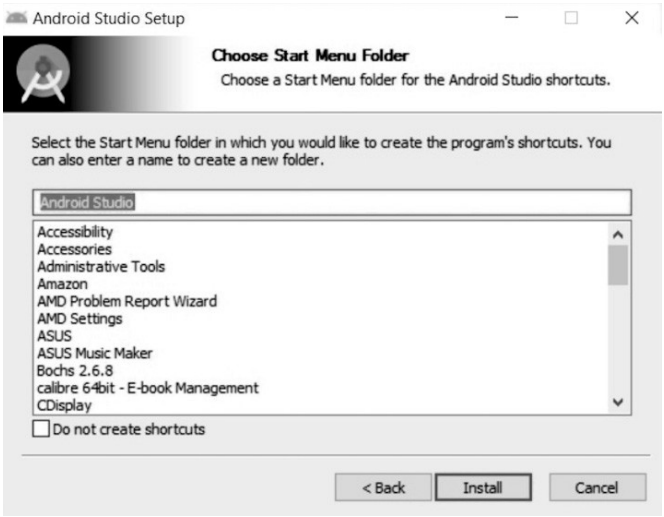


Slika 3.3, odabir komponenti za instalaciju

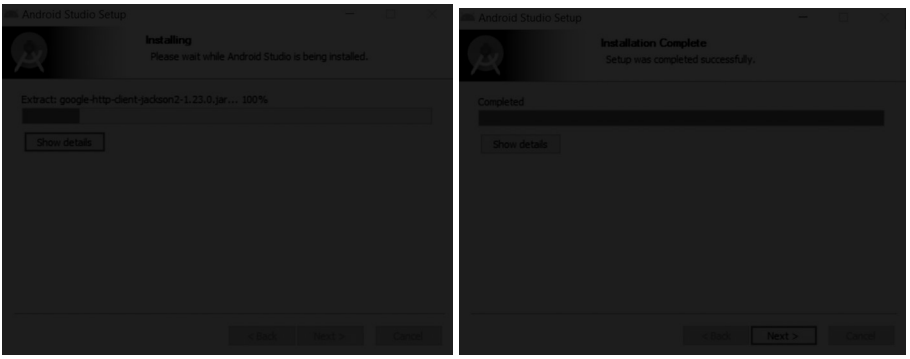
Sledeći panel, pod nazivom Configuration Settings, specificira lokaciju na kojoj će Android Studio biti instaliran (slika 3.4). Moguće je ostaviti podrazumevanu lokaciju, ili specificirati drugu. U svakom slučaju, specificirana lokacija mora imati bar 500 MB slobodnog prostora kako bi instalacija mogla biti izvršena. U sledećem meniju moguće je odabrati Start Menu direktorijum gde će biti smeštena prečica okruženja, kao što je prikazano na slici 3.5. Mogu se ostaviti podrazumevana podešavanja i kliknuti dugme Next, što započinje proces instalacije, kao što je prikazano na slici 3.6.



Slika 3.4, specifikacija lokacije gde će Android Studio biti instaliran



Slika 3.5, odabir Start Menu direktorijuma

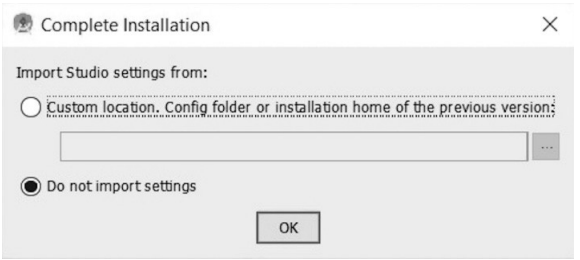


Slika 3.6, proces instalacije Android Studio okruženja

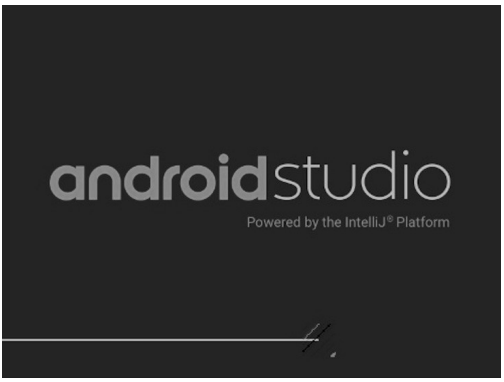
U toku instalacije moguće je kliknuti Show Details dugme, koje će prikazati detaljno koja se komponenta ili fajl trenutno instalira, kao i druge aktivnosti samog procesa instalacije. Kada se instalacija završi, pojaviće se panel Completing Android Studio Setup, kao što je prikazano na slici 3.7. Time se završava proces instalacije okruženja.

Slika 3.7, završetak instalacije Android Studio okruženja

Prilikom prvog pokretanja Android Studio okruženja, prikazaće se Complete Installation dijalog, koji nudi opciju importa podešavanja od prethodne instalacije koja je bila instalirana na tom računaru (ukoliko postoji), kao što je prikazano na slici 3.8. Ukoliko je ovo prva instalacija Android Studio okruženja na tom računaru, ili ukoliko se ne želi import podešavanja od prethodne verzije, bira se opcija Do not import settings. Nakon potvrde, počeće pokretanje okruženja, kao što je prikazano na slici 3.9.

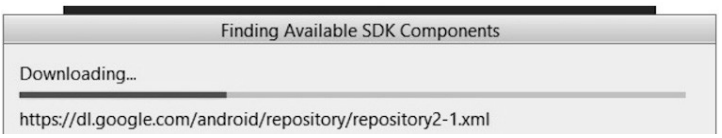


Slika 3.8, import podešavanja od prethodne verzije (ukoliko postoji)



Slika 3.9, Android Studio splash screen

Prilikom prvog učitavanja okruženja, moguće je da će Android Studio u ovom trenutku početi da preuzima sa Interneta potrebne biblioteke, kao što je prikazano na slici 3.10. To su SDK komponente koje su dostupne i neophodne



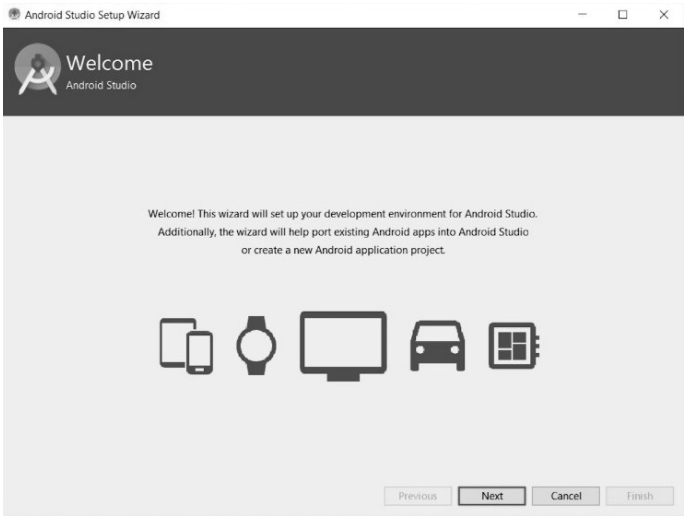
okruženju.

Slika 3.10, preuzimanje potrebnih SDK komponenti prilikom prvog pokretanja okruženja

Proces preuzimanja potrebnih SDK komponenti može da potraje nekoliko minuta u zavisnosti od brzine Internet konekcije, pa zbog toga prvo pokretanje okruženja traje malo duže u poređenju sa svakim sledećim pokretanjem. Kada su sve potrebne biblioteke preuzete, prikazaće se Android Studio ekran dobrodošlice, kao na slici 3.11. Ovaj ekran obaveštava korisnika da će čarobnjak sada podesiti okruženje za razvoj, i eventualno pomoći u portovanju već postojećih Android aplikacija (razvijanim u starijim verzijama okruženja) ili

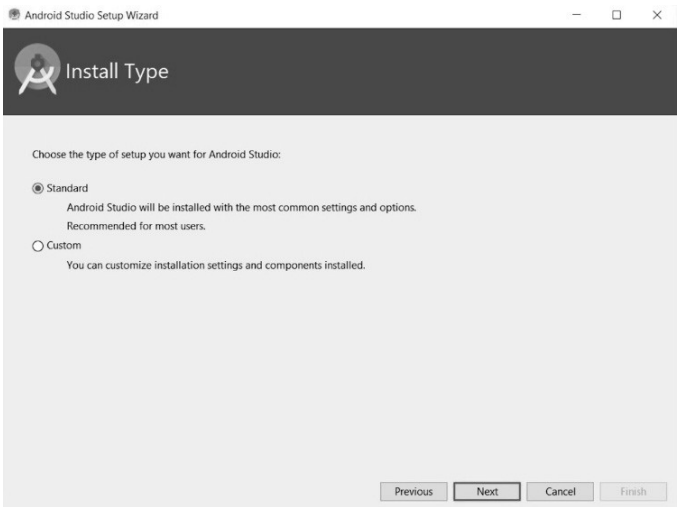
pokrenuti novi projekat sa novom Android aplikacijom. Na ekranu su grafički prikazane i sve različite

platforme koje su podržane od strane okruženja i za koje je moguće razvijati aplikacije (Android telefoni, satovi, TV i slično).

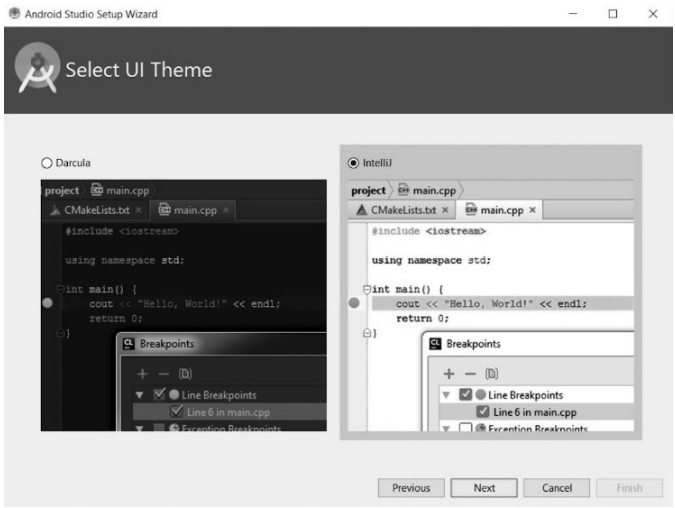


Slika 3.11, Android Studio čarobnjak za podešavanje okruženja

Sledeća dva ekrana traže od korisnika da odabere tip instalacije (podrazumevana opcija u potpunosti odgovara), kao što je prikazano na slici 3.12, kao i opciju da se odabere tema korisničkog interfejsa okruženja, kao što je prikazano na slici
3.13. Ponuđene su dve teme, tamna (Darcula) i standardna svetla (IntelliJ), sa uzorkom prikaza obe teme. Odabir teme je u potpunosti individualan i ostavljen korisniku, pošto razlike u funkcionalnosti nema.

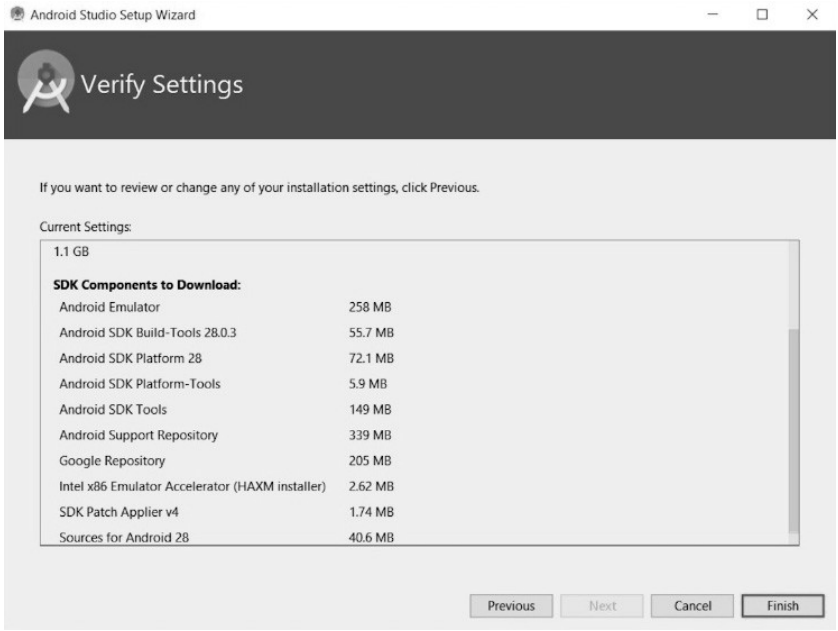


Slika 3.12, odabir tipa instalacije okruženja

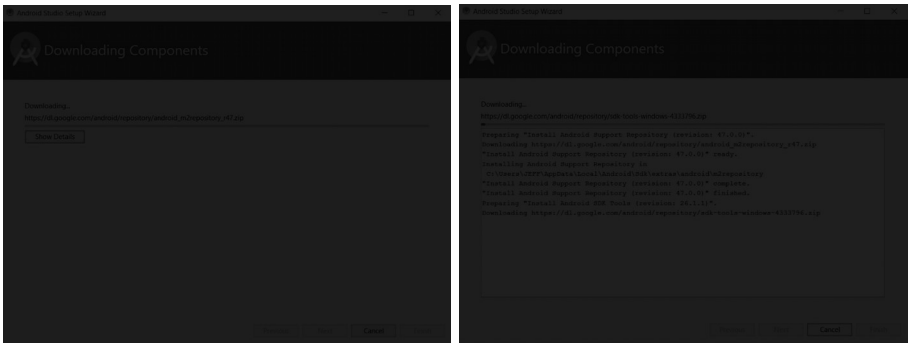


Slika 3.13, odabir teme korisničkog interfejsa okruženja

Nakon odabira tipa instalacije i teme, Android Studio nudi priliku da se verifikuju sva odabrana podešavanja i opcije, kao što je prikazano na slici 3.14. Na ovom ekranu su prikazane i sve dodatne komponente koje su identifikovane kao potrebne i koje će Android Studio početi da preuzima i instalira nakon potvrde opcija i klika na Finish dugme.

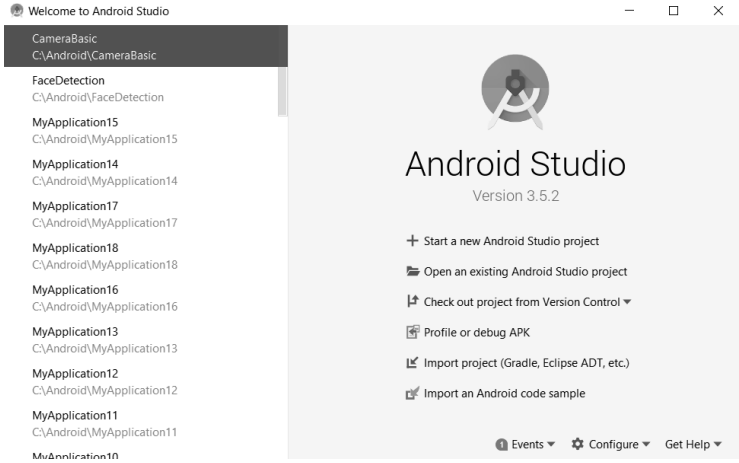


Slika 3.14, verifikacija odabranih opcija



Slika 3.15, čarobnjak preuzima i instalira SDK komponente, sa i bez prikaza detalja

Nakon verifikacije odabranih komponenti, čarobnjak počinje da preuzima, otpakuje i instalira sve neophodne SDK komponente. Ovaj proces može da potraje i desetak minuta, pa se dosada prilikom čekanja može delimično ublažiti klikom na Show Details. Na taj način korisnik može videti koji se svi različiti fajlovi trenutno preuzimaju, njihov proces otpakivanja, kao i lokaciju gde se instaliraju. Kada se konačno preuzmu sve neophodne komponente, prikazaće se ekran dobrodošlice kao na slici 3.16. Ovaj dijalog se može iskoristiti za pokretanje novog Android projekta, otvaranje već postojećeg projekta, kao i opcije za checkout projekta sa nekog alata za kontrolu verzije (npr. GIT), odnosno import projekta. Ukoliko ovo nije prvo pokretanje okruženja, prethodni projekti će biti izlistani sa leve strane ekrana, kao što je prikazano na slici 3.16.



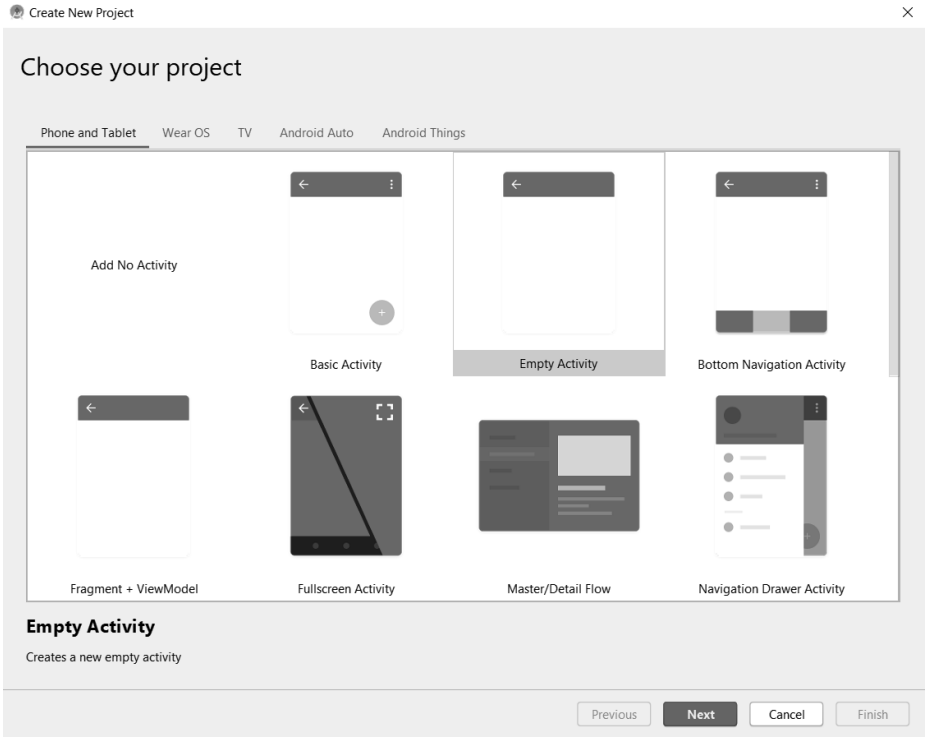
Slika 3.16, Welcome to Android Studio ekran

Strukturu i korisnički interfejs Android Studio okruženja je najlakše opisati kroz kreiranje prvog projekta, odnosno jednostavne Android aplikacije.

3.2. Kreiranje i struktura projekta

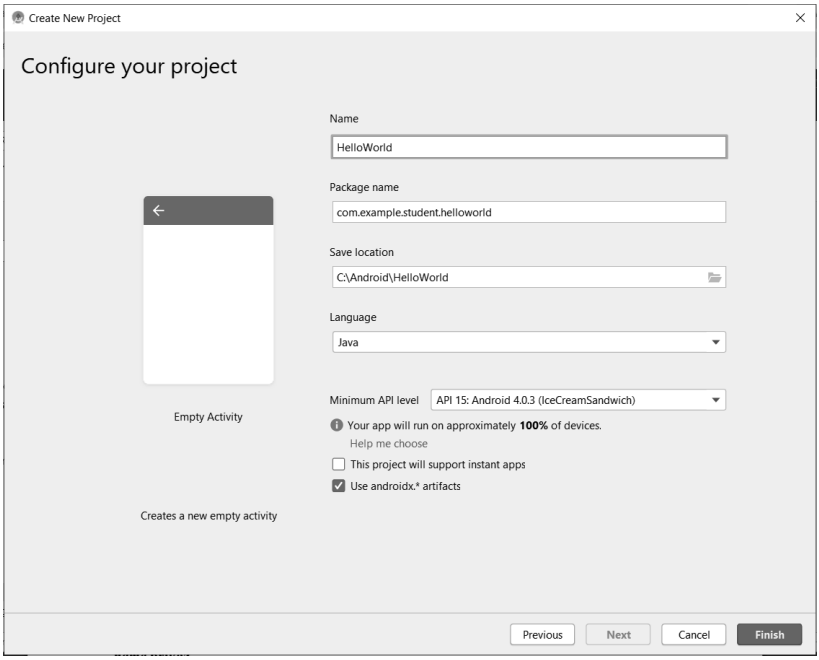
U ovom poglavlju opisan je proces kreiranja Android projekta, kroz koji ćemo objasniti osnovne alate koji se koriste u programiranju Android aplikacija. Prvi program koji se piše u bilo kom novom programskom jeziku ili okruženju obično je jednostavna „Hello World“ aplikacija, koju ćemo u ovom slučaju iskoristiti za opis korisničkog interfejsa okruženja. Nakon koraka opisanih u poglavlju 3.1, došli smo do ekrana sa slike 3.16 (Welcome to Android Studio ekran). Odabirom opcije **Start a new Android Studio project** započinje proces kreiranja novog projekta (ovaj proces, kao i redosled i sadržaj ekrana za konfiguraciju projekta se može malo razlikovati od verzije do verzije Android Studio okruženja). Proces je prikazan na verziji Android Studio 3.5.2.

Sledeći ekran koji se prikazuje nudi opcije za odabir platforme za koju se kreira aplikacija, kao i šablon na osnovu kojeg će biti kreirana početna aktivnost, kao što je prikazano na slici 3.17. Kao platforma u ovom primeru bira se telefon (prva opcija - **Phone and Tablet**), a kao šablon koristi se prazna aktivnost (**Empty Activity**).



Slika 3.17, odabir platforme i šablona aktivnosti za novi projekat

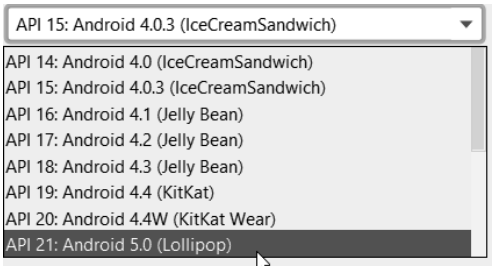
Nakon odabira platforme i šablona, otvara se forma za podešavanje projekta, kao što je prikazano na slici 3.18. Proces započinje unošenjem imena aplikacije u polje **Name**. Ime aplikacije će istovremeno služiti i kao ime samog projekta. Treba obratiti pažnju da se menjanjem imena aplikacije automatski menja i ime paketa koje će se koristiti u okviru projekta (polje sa imenom **Package name**). U slučaju da je potrebno drugačije ime paketa, može se naravno ručno izmeniti. Sledeće polje se odnosi na lokaciju na kojoj će se projekat čuvati (polje **Save location**), gde se može uneti željena putanja. U sledećem polju (polje sa imenom **Language**) bira se programski jezik u kome će se razvijati aplikacija. Dostupne opcije su Java i Kotlin. U okviru ovog udžbenika biće korišćena Java kao odabrani programski jezik.



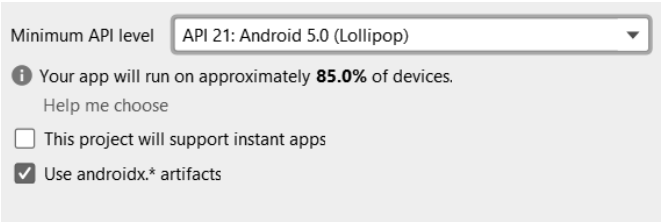
Slika 3.18, osnovno podešavanje projekta

Veoma bitna opcija je minimalni API nivo (dropdown lista **Minimum API level**). Ova vrednost označava minimalni API nivo na kojem će biti moguće instalirati i pokrenuti aplikaciju, pošto će uređaji sa nižim nivoom od zadatog minimalnog nivoa odbiti instalaciju aplikacije. Dodatno, Android Studio daje i statistički podatak koji opisuje koliko procenata mobilnih telefona na tržištu će moći da pokrene aplikaciju. Kao što se vidi sa slike 3.18, ukoliko se kao minimalni API nivo odabere Android 4.0.3 (Ice Cream Sandwich), broj podržanih uređaja je 100%, odnosno aplikacija će moći da se pokrene na praktično svim aktivnim Android uređajima. U slučaju odabira više verzije minimalnog API nivoa (slika 3.19), manji procenat uređaja će moći da pokrene

aplikaciju (slika 3.20). U ovom primeru koristićemo Android 5.0 Lollipop kao minimalni API nivo, odnosno aplikacija će moći da se pokrene na približno 85% aktivnih uređaja. Treba znati da što je viša vrednost minimalnog API nivoa, procenat uređaja drastično opada, tako da u slučaju odabira Android 9.0 (Pie) ovaj procenat je manji od 1%.

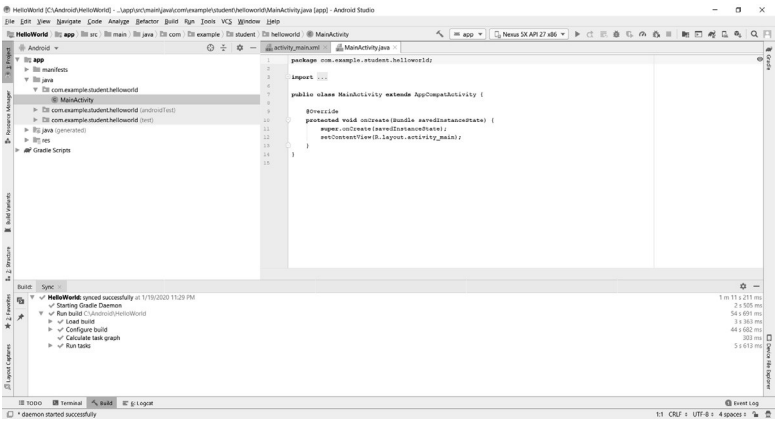


Slika 3.19, odabir druge vrednosti minimalnog podržanog API nivoa



Slika 3.20, procenat uređaja koji će moći da pokrenu aplikaciju na osnovu zadatog minimalnog API nivoa

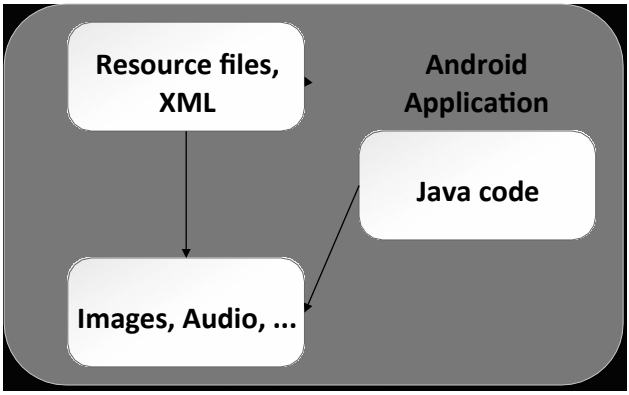
Nakon potvrde izabranih opcija klikom na dugme **Finish**, otvara se okruženje sa projektom pripremljenim prema prethodno odabranoj konfiguraciji, kao na slici 3.21.



Slika 3.21, kreirani HelloWorld projekat

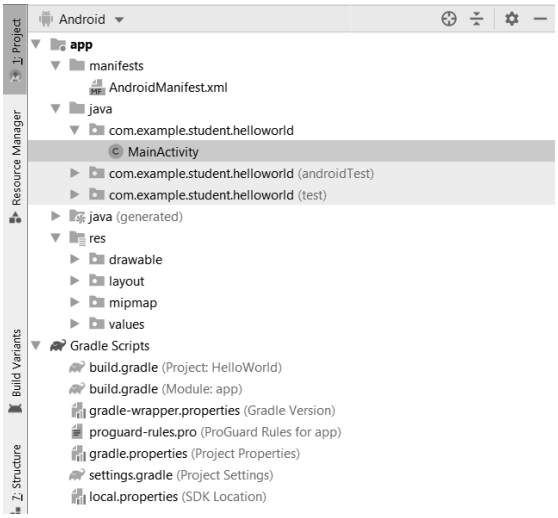
Treba još napomenuti da, ukoliko SDK za odabranu verziju Androida nije instaliran, Android studio može u tom slučaju početi da skida potrebne biblioteke.

Glavni princip razvoja mobilnih Android aplikacija leži u činjenici da se kod jasno razdvoji od resursa. Zbog toga se svaka Android aplikacija sastoji iz programskog koda i resursa, u koje spadaju XML fajlovi kao i drugi dodatni fajlovi poput slika, audio i video materijala i slično (slika 3.22).



Slika 3.22, stuktura standardne Android aplikacije

Na sličan način, svaki projekat u Android Studio okruženju sadrži jedan ili više modula koji sadrže izvorni kod i resursne fajlove. Podrazumevano, Android Studio prikazuje sve projektne fajlove u Android pogledu projekta, kao što je prikazano na slici 3.23. Ovaj pogled je dodatno organizovan po modulima, kako bi se omogućio brz pristup ključnim projektnim fajlovima.



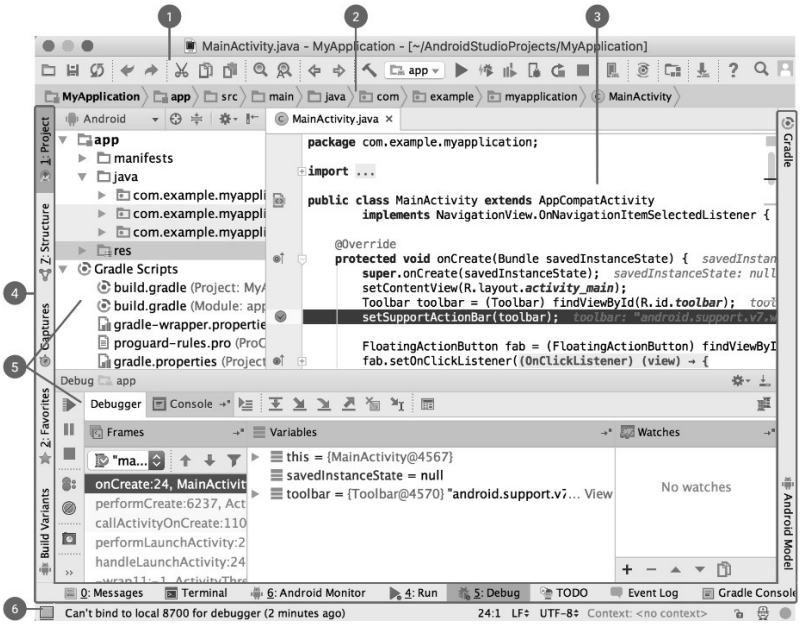
Slika 3.23, projektni fajlovi u Android pogledu

Aplikacioni modul (**app**) sadrži sledeće foldere:

- **manifests** – sadrži AndroidManifest.xml fajl.
- **java –** sadrži izvorne fajlove sa Java kodom, uključujući i JUnit testove.
- **res –** sadrži sve što nije kod, poput XML rasporeda komponenti (**layout**), korisničkih stringova, dimenzija, boja, tema i bitmap slika.

Modul **Gradle Scripts** sadrži gradle fajlove (konfiguracione fajlove Android projekta). Njih nije potrebno ručno pisati, pošto se automatski generišu pri kreiranju projekta, mada se po potrebi njihov sadržaj može menjati u toku razvoja same aplikacije. Gradle fajlovi sadrže podatke poput poput minimalne verzije API nivoa, ID aplikacije ili opisa eksternih biblioteka koje su potrebne aplikaciji. Gradle se može koristiti i van Android Studija, za potrebe drugih projekata koji su pisani drugačijim programskim jezicima (poput Maven-a), odnosno nije isključivo alat koji se koriti samo u okviru Android Studio okruženja. Ukoliko je čitalac upoznat sa Maven alatom, videće da postoje velike sličnosti između dva alata.

Sam glavni prozor Android Studio okruženja se sastoji od više logičkih celina, označenih na slici 3.24 (slika preuzeta sa Google – developer.android veb sajta). Programer može organizovati prikaz glavnog ekrana kako bi ga dodatno prilagodio sebi (u okviru ovog udžbenika mi ćemo koristiti podrazumevani prikaz glavnog ekrana).



Slika 3.24, struktura glavnog prozora Android Studio okruženja (izvor: https://developer.android.com/studio/intro/)

Svaka od označenih celina ima svoju svrhu:

1. **Linija sa alatima (toolbar)** sadrži veliki opseg različitih akcija koje se mogu izvršiti, uključujući pokretanje izvršavanja aplikacije i pokretanje različitih Android alata.
2. **Navigaciona linija (navigation bar)** služi za navigaciju kroz projekat, kao i otvaranje fajlova za editovanje. Prikazuje apsolutnu putanju do fajla koji se nalazi fokusiran u editoru. Pruža mnogo kompaktniji pogled od strukture koja se vidi pod **Project** prozorom.
3. **Editor** je celina gde se piše i modifikuje kod. U zavisnosti od konkretnog tipa fajla koji je trenutno otvoren, sam editor se može promeniti. Na primer, kada se posmatra XML fajl razmeštaja komponenti, editor će prikazati grafički pregled trenutnog sadržaja XML fajla.
4. **Linija sa prozorima alata (tool window bar)** se nalazi po obodu ekrana okruženja, sadrži prečice pomoću kojih se može prikazati ili sakriti pojedinačni prozor sa konkretnim alatom.
5. **Prozor sa alatima (tool window)** daje pristup debageru, kao i posebnim zadacima poput pretrage, kontrole verzije i slično. Svaki od ovih alata pojedinačno se može prikazati ili sakriti.
6. **Statusna linija (status bar)** prikazuje status projekta i samog okruženja, uz eventualna upozorenja ili poruke.

Pogledajmo sada detaljnije sadržaj koji je Android Studio generisao za potrebe našeg projekta. U editoru možemo videti dva taba, jedan pod nazivom MainActivity.java, a drugi pod nazivom activity_main.xml. MainActivity.java je Java klasa koja predstavlja ulaznu tačku u našu aplikaciju. U pitanju je aktivnost, koja predstavlja jednu od četiri osnovne Android komponente. Aktivnost predstavlja osnovni blok korisničkog interfejsa Android aplikacije, analogna je jednoj stranici tradicionalne veb aplikacije. Aplikacija obično ima više od jedne aktivnosti, a korisnik aplikacije se kreće kroz njih na sličan način kao kada se koristi veb aplikacija sa više strana.

Prilikom programiranja Android aplikacija, potrebne klase se izvode iz predefinisanih Android baznih klasa, što se može uočiti i na našem primeru, jer je MainActivity izvedena iz Androidove klase AppCompatActivity. Automatski generisani sadržaj ove klase dat je sa:

```
package com.example.student.helloworld;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

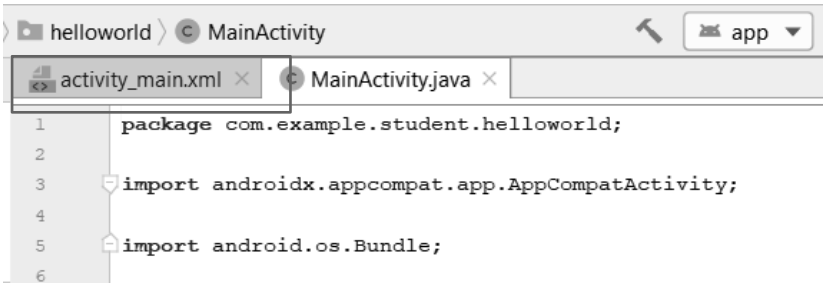
```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Može se uočiti da postoji samo jedna metoda, onCreate(), koja predstavlja nadjačanu metodu (anotacija @Override) iz bazne klase AppCompatActivity. Ono što je potrebno razumeti za sada jeste da ova metoda, pozivom metode setContentView() postavlja na ekran sadržaj aktivnosti koji je definisan u XML fajlu prosleđenom kao parametar ove metode – R.layout.activity_main.

Svaka aktivnost, osim Java koda, treba da ima i definiciju vizuelnog izgleda korisničkog interfejsa koji odgovara toj aktivnosti. Kao što smo ranije naveli, resursi (među koje spada i XML fajl sa rasporedom komponenti na ekranu – layout fajl) su potpuno odvojeni od Java koda, i nalaze se u folderu res/layout. Poziv metode setContentView() zapravo povezuje aktivnost i njen XML fajl sa rasporedom komponenti, u kome su definisane:

- sve komponente koje su potrebne na ekranu.
- atributi koji definišu vizuelni prikaz tih komponenti.
- raspored komponenti na ekranu.

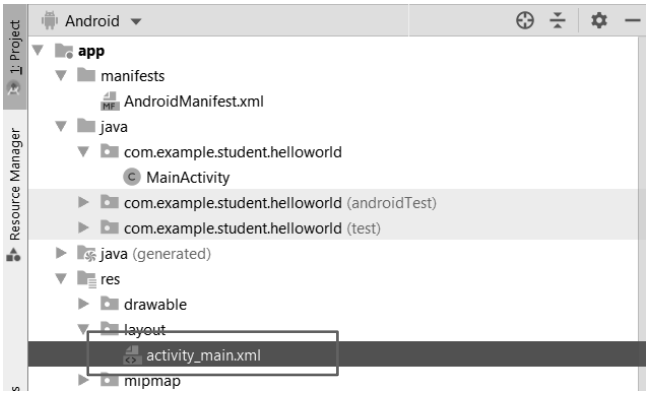
Jednoj Activity klasi se pridružuje jedan fajl sa rasporedom komponenti. Po konvenciji, ime XML fajla sa rasporedom treba da odgovara imenu same Activity klase na sledeći način – ako se aktivnost zove MainActivity, njen raspored se zove activity_main.xml. Na isti način je i okruženje automatski imenovalo naš raspored komponenti u primeru. Inače, konvencija imenovanja koja važi u programskom jeziku Java važi i u slučaju Android programiranja. Imena klasa se drže standardne Camel Case notacije (prvo slovo svake reči u imenu veliko, ostala slova mala), što se i vidi u slučaju imena Activity klase – MainActivity. Na prikaz XML fajla se dolazi klikom na tab u editoru sa imenom activity_main.xml, kao što je prikazano na slici 3.25.



Slika 3.25, tab sa XML rasporedom komponenti u editoru

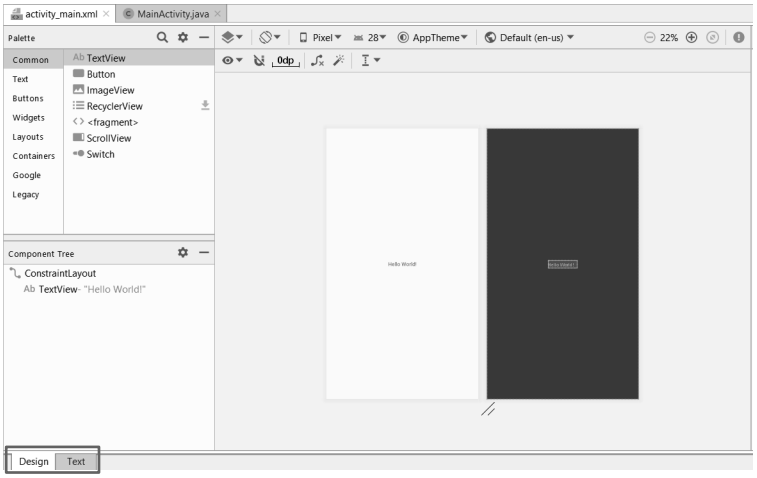
Ukoliko XML fajl nije učitan u editoru, može mu se pristupiti i kroz strukturu projekta. Kao što je navedeno ranije, XML fajlovi sa rasporedima se nalaze u

res/layout folderu, gde možemo pronaći i traženi raspored, kao što je prikazano na slici 3.26. Pošto jedna aplikacija može imati više aktivnosti, u ovom folderu se u opštem slučaju može nalaziti više fajlova sa rasporedima komponenti.



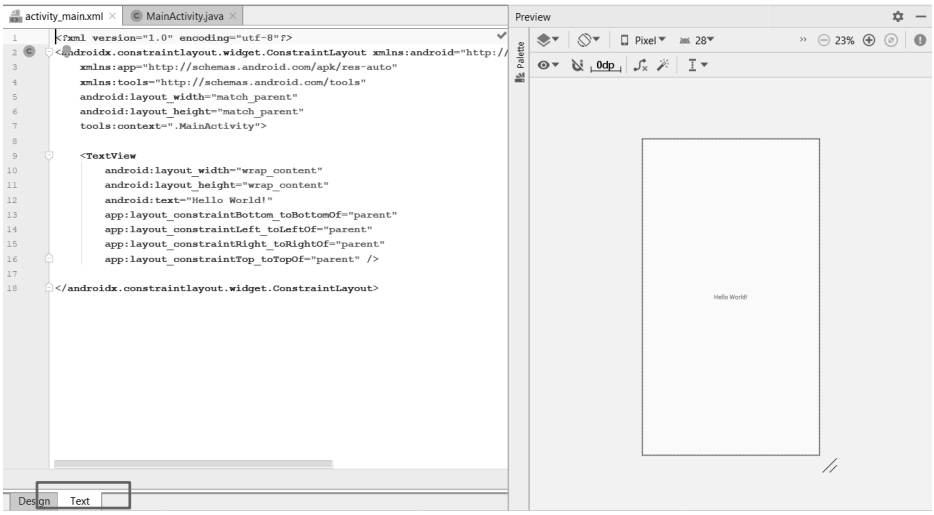
Slika 3.26, lokacija XML fajlova sa rasporedom u strukturi projekta (res/layout)

Nakon otvaranja fajla sa rasporedom komponenti, podrazumevano će se otvoriti ekran za grafičkim editorom za raspored komponenti. Ovaj editor omogućava veoma jednostavno kreiranje grafičkog interfejsa, uz podršku za drag & drop komponenti grafičkog interfejsa iz palete dostupnih komponenti. Svaka promena se trenutno prikazuje i vizuelno, kao što je prikazano na slici 3.27. U našem slučaju, automatski je generisan ekran sa jednom tekstualnom labelom (tip **TextView**), čiji je sadržaj string „Hello World!“. U narednim poglavljima ovog udžbenika biće prikazano kako se postavljaju nove komponente u raspored i kako se definišu njihovi vizuelni atributi. U ovom primeru, ostavićemo automatski generisani sadržaj.



Slika 3.27, grafički editor za XML fajlove sa rasporedom komponenti

Na slici 3.27, crvenom bojom su označena dva taba na dnu editora. Prvi tab, pod imenom **Design**, je podrazumevano selektovan, i omogućava vizuelni prikaz XML fajla. Drugi tab, sa imenom **Text**, omogućava pregled sadržaja samog XML fajla. Odabirom ovog taba, dobija se XML kod koji odgovara grafičkom prikazu iz grafičkog editora, kao što je prikazano na slici 3.28.



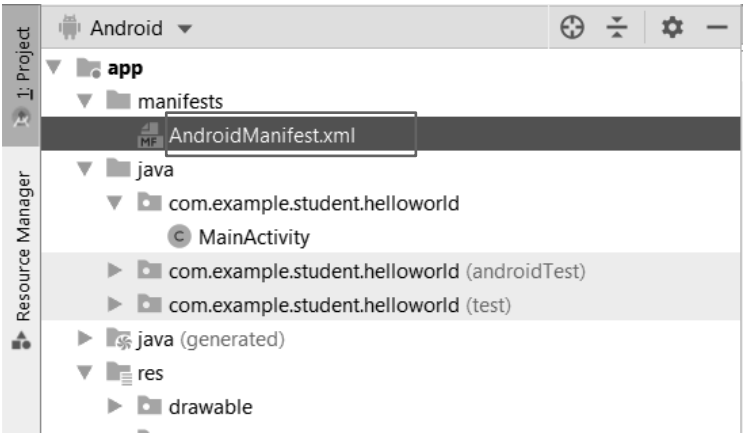
Slika 3.28, pregled sadržaja XML koda rasporeda komponenti

Izdvojen sadržaj generisanog fajla je dat u nastavku:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Iz datog koda XML fajla vidimo na koji način je definisano polje tipa TextView koje sadrži tekst „Hello World!“. Ovaj fajl definiše sadržaj ekrana naše aktivnosti MainActivity, i on će biti prikazan kada se pokrene aplikacija i kada se pokrene onCreate metoda naše aktivnosti.

Pogledajmo sada i sadržaj ostalih bitnih fajlova aplikacije. Kao što smo naveli, AndroidManifest.xml je jedan od najznačajnijih fajlova aplikacije, u kome se nalazi opis ključnih informacija o aplikaciji koje su potrebne Android build alatima i samom Android operativnom sistemu. To su informacije poput imena paketa, svih komponenti aplikacije (aktivnosti, servisi, itd.), kao i eventualnih dozvola koje su potrebne aplikaciji za rad. Fajl se može pronaći u folderu manifests, kao što je prikazano na slici 3.29.

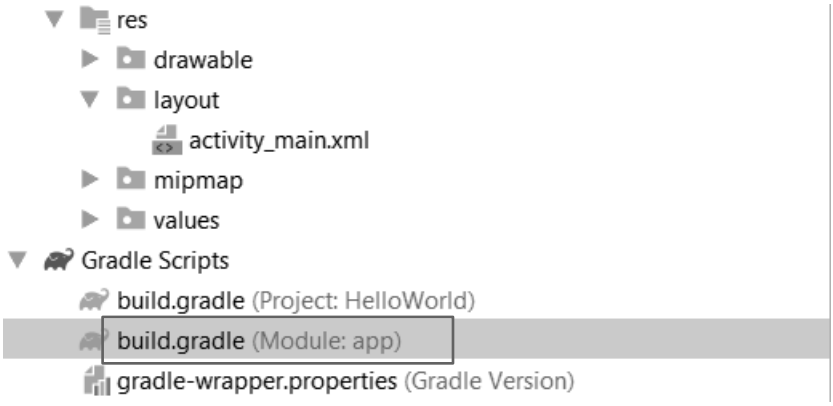


Slika 3.29, lokacija AndroidManifest.xml fajla u strukturi projekta

Primer sadržaja AndroidManifest.xml fajla dat je u nastavku:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.student.helloworld">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Pogledajmo još za kraj šta se nalazi u build.gradle fajlu (Module:app). Njemu se može pristupiti takođe iz strukture samog projekta, kao što je prikazano na slici 3.30. Ovaj fajl se nalazi u okviru foldera Gradle Scripts, zajedno sa drugim skriptama koje su neophodne za uspešno build-ovanje aplikacije.



Slika 3.30, build.gradle fajl za modul app i njegova lokacija u strukturi projekta

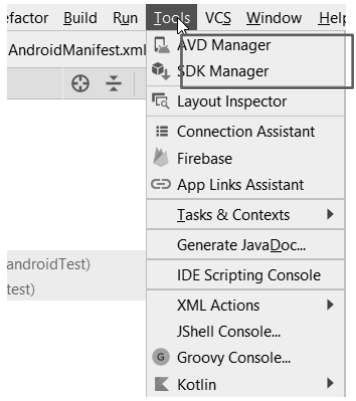
Otvaranje ovog fajla će u editoru prikazati sadržaj same skripte, kao što je prikazano na slici 3.31. Ovde je moguće videti (i izmeniti) targetSdkVersion, koja je u ovom slučaju postavljena na vrednost 28. To znači da se kompajliranje aplikacije koristi verzija biblioteka Androida koja se nalazi u API nivo 28. Ovde je definisana i minSdkVersion, koja označava minimalnu verziju API nivoa na kojoj će biti moguće pokrenuti aplikaciju, i ona je u ovom projektu postavljena na API nivo 21 (Lollipop). U ovom fajlu je moguće videti i ID aplikacije, kao i njenu verziju. U sekciji **dependencies** mogu se videti sve eksterne biblioteke koje su podrazumevano uključene (naravno, moguće je uključiti i dodatne biblioteke po potrebi, na sličan način kao što je prikazano na slici). U našem primeru, ostavićemo podrazumevana podešavanja koja su automatski generisana prilikom kreiranja projekta.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.example.student.helloworld"
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
}
```

Slika 3.31, sadržaj build.gradle fajla za modul app

3.3. Konfigurisanje Emulatora i pokretanje aplikacije

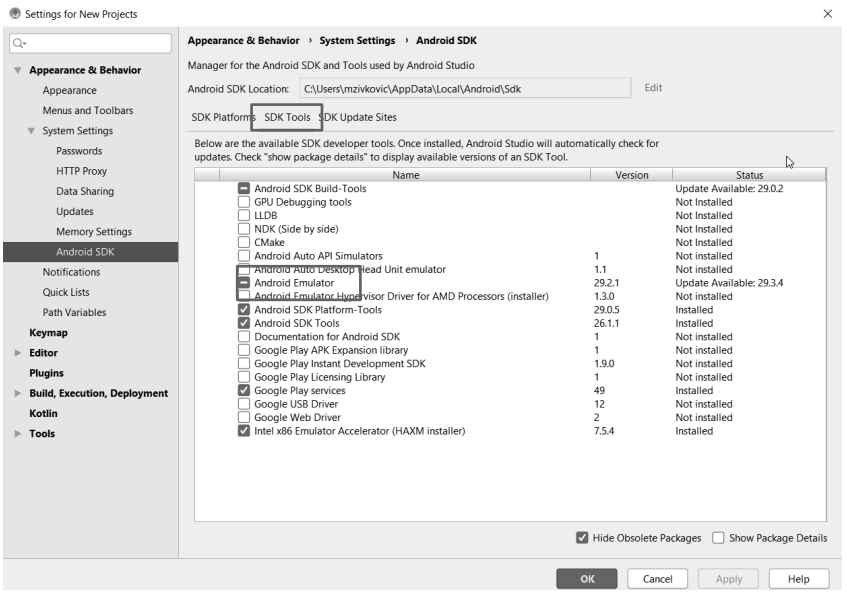
Aplikacija kreirana u prethodnom poglavlju je u potpunosti funkcionalna. Kako bismo je pokrenuli, neophodno je da konfigurišemo **Android Emulator**. Već smo opisali da Emulator nudi skoro sve funkcionalnosti pravog, fizičkog uređaja. Moguće je simulirati veliki broj korisničkih scenarija, poput dolaznih poziva, lokacije uređaja, različitih performansi mobilne mreže, različitog nivoa baterije i slično. Postoji nekoliko stvari koje nije moguće simulirati putem virtuelnog uređaja, poput Bluetooth, NFC, slušalica prikačenih za uređaj i USB



konekcije.

Slika 3.32, sekcija Tools, gde se nalaze AVD i SDK Manager opcije

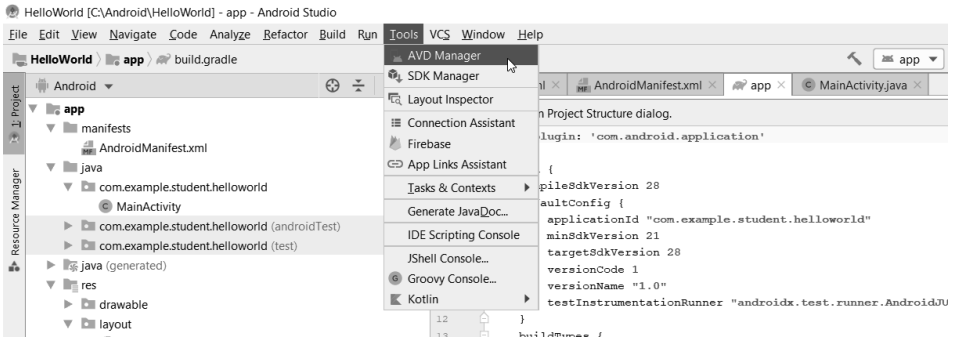
Za instalaciju Android Emulatora, potrebno je odabrati sekciju **Tools**, kao što je prikazano na slici 3.32, zatim **SDK Manager** opciju, a nakon toga u sledećem ekranu odabrati tab **SDK Tools**, kao što je prikazano na slici 3.33.



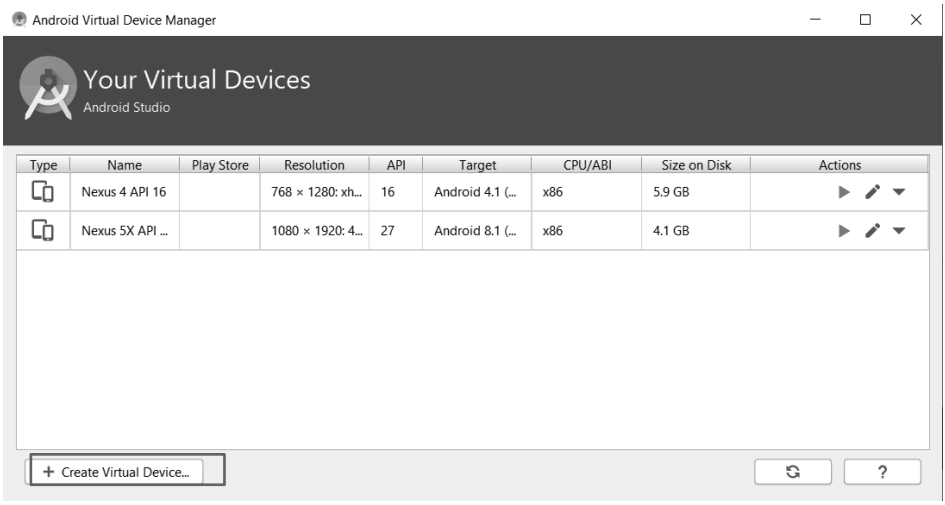
Slika 3.33, SDK Tools tab u okviru SDK Manager sekcije

Za računare sa Intel procesorom, preporučljivo je instalirati i opciju **Intel HAXM** (Intel Hardware Accelerated Execution Manager) selektovanjem poslednje opcije u listi sa slike 3.33. Na ovaj način se putem funkcije hardverskog ubrzavanja drastično mogu poboljšati performanse Emulatora kroz ubrzanje virtuelne mašine. Kako bi virtuelizacija radila, opcija virtuelizacije mora biti aktivirana kroz BIOS računara.

Svaka instanca Android Emulatora koristi Android virtuelni uređaj (engl. Android Virtual Device) kako bi specificirala verziju Androida i hardversku konfiguraciju simuliranog uređaja. Nakon instalacije Emulatora, moguće je dodati različite Android virtuelne uređaje (AVD) kroz **AVD Manager**, koji se aktivira kroz sekciju Tools, kao što je prikazano na slici 3.34. Nakon ulaska u **AVD Manager**, prikazuju se svi raspoloživi virtuelni uređaji (sa opcijama pokretanja virtuelnog uređaja, modifikacije i slično), kao i opcija za dodavanje novog virtuelnog uređaja, kao što je prikazano na slici 3.35. U praksi, preporučljivo je dodati AVD za svaku konfiguraciju koju aplikacija treba da podrži, na osnovu podešavanja <uses-sdk> u okviru AndroidManifest.xml fajla.



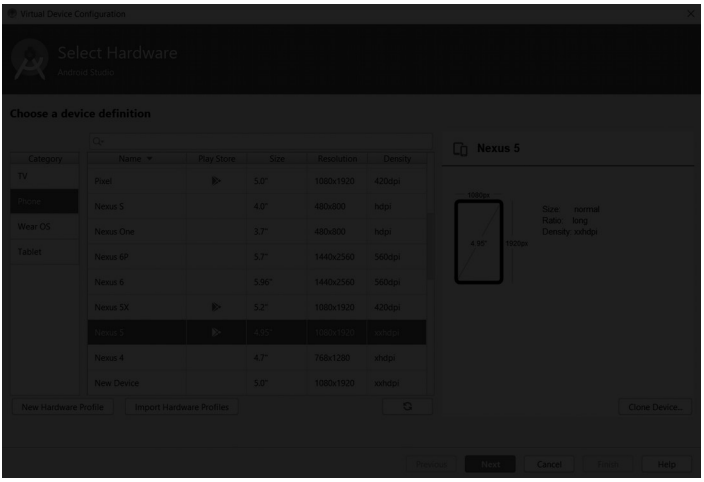
Slika 3.34, AVD Manager u okviru sekcije Tools



Slika 3.35, Raspoloživi virtuelni uređaji u okviru opcije AVD Manager

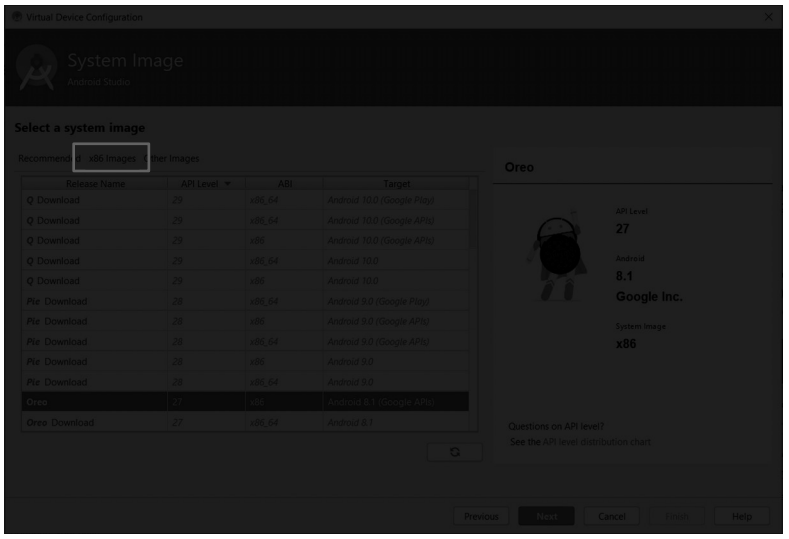
Svaki AVD sadrži hardverski profil, sistemska slika (engl. *system image*), memoriju za smeštanje podataka i druga svojstva. Hardverski profil definiše skup karakteristika fizičkog uređaja. **AVD Manager** dolazi sa predefinisanim hardverskim profilima (poput Pixel i Nexus uređaja). Prilikom kreiranja novog virtuelnog uređaja, prvo je potrebno odabrati hardverski profil, kao što je prikazano na slici 3.36. Treba obratiti pažnju da su samo neki uređaji označeni da uključuju **Play Store**. To su tzv. CTS komercijalno usaglašeni uređaji (Compability Test Suite – CTS), koji mogu koristiti odgovarajuće sistemske verzije i imaju oznaku u koloni Play Store. U ovom primeru kreiraćemo novi Nexus 5 uređaj.

Kada je odabran hardverski profil, prelazi se na sledeći ekran gde je potrebno odabrati sistemsku sliku. U slučaju da se koristi Intel hardverska akceleracija, potrebno je odabrati **x86 Images** tab, kako bi se dobile što bolje performanse.



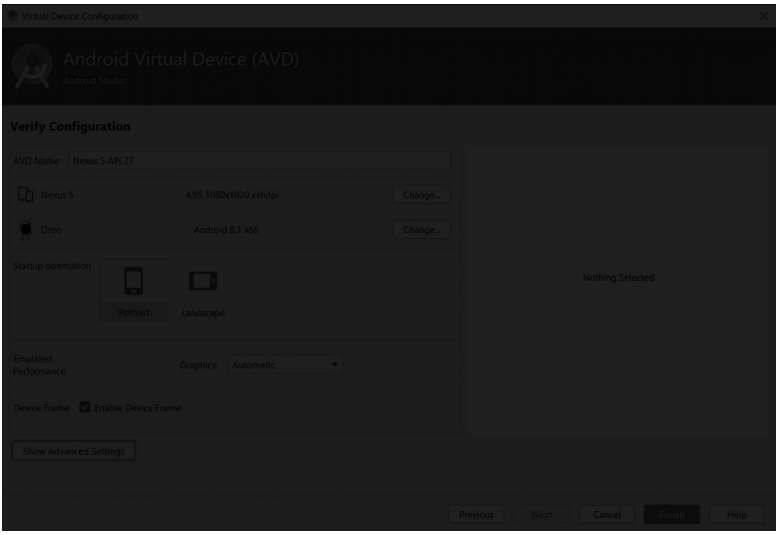
Slika 3.36, odabir hardverskog profila novog virtuelnog uređaja

Neke verzije slika imaju dodato **Google APIs** u nazivu, što označava mogućnost pristupima Google Play servisima. Odabir sistemske slike prikazan je na slici
3.37. Za potrebe ovog primera odabraćemo Android Oreo 8.1 API nivo 27 sa opcijom Google APIs. Međutim, za potrebe vežbanja, naročito na slabijim računarima (manje od 8 GB RAM), preporučljivo je uzeti neki slabiji hardverski profil (npr. Nexus 4) sa nešto starijom verzijom Androida (na primer Jelly Bean 4.1 API nivo 16), zbog znatno bržeg i komfornijeg rada i boljih performansi emulatora. Ukoliko pored oznake Android verzije piše **Download**, potrebno je prvo preuzeti sistemsku sliku pre nastavka dalje.

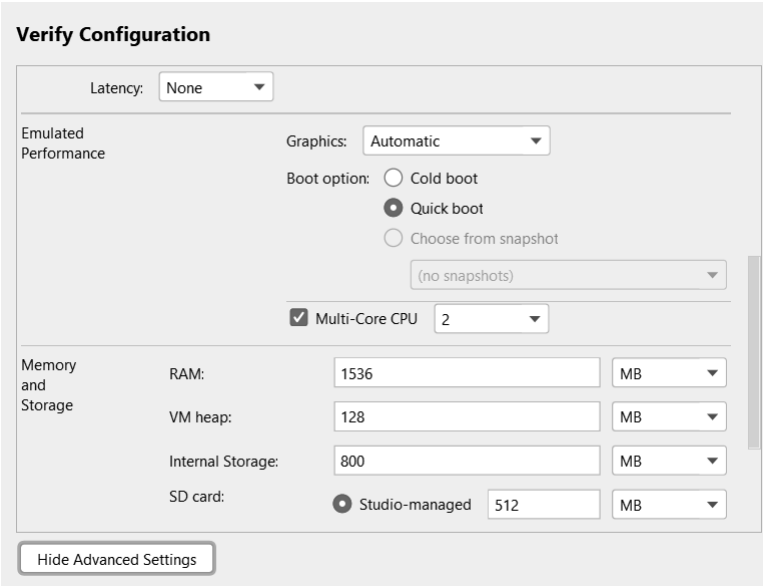


Slika 3.37, odabir sistemske slike virtuelnog uređaja

Potvrdom se prelazi na potvrdu odabrane konfiguracije. Moguće je dati posebno ime virtuelnoj mašini, odabrati orijentaciju uređaja nakon pokretanja emulatora, kao i prikazati napredna podešavanja (slika 3.39), gde je moguće odabrati skin, postaviti količinu memorije koju će uređaj koristiti, podesiti prednju i zadnju kameru i slično.

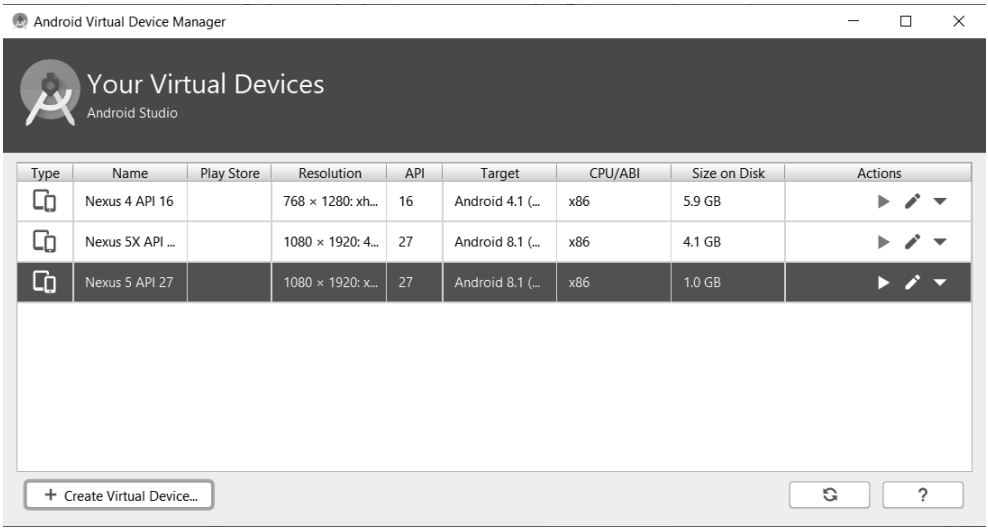


Slika 3.38, verifikacija odabrane konfiguracije



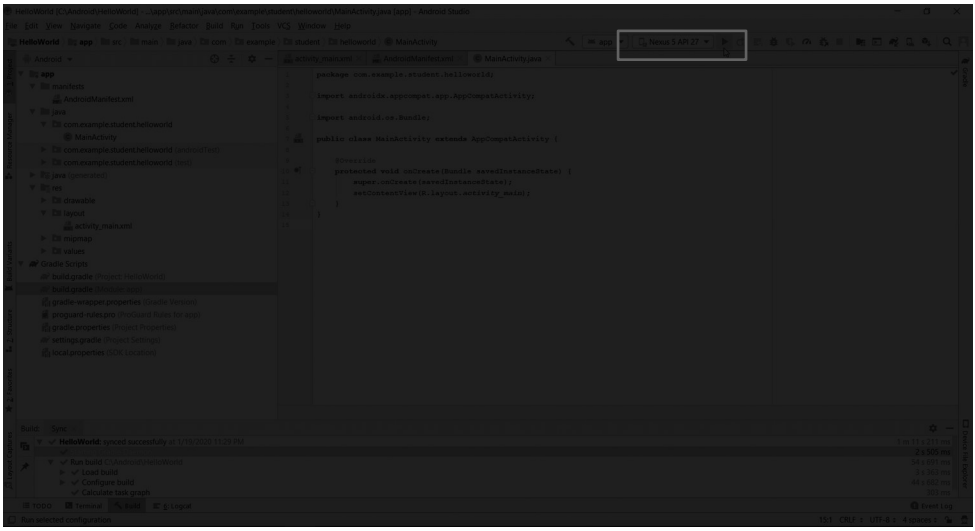
Slika 3.39, napredna podešavanja virtuelnog uređaja

Klikom na dugme **Finish** na slici 3.38 kreira se željeni virtuelni uređaj. Odmah nakon kreiranja, uređaj je vidljiv u **AVD Manager** ekranu, kao što je prikazano na slici 3.40.



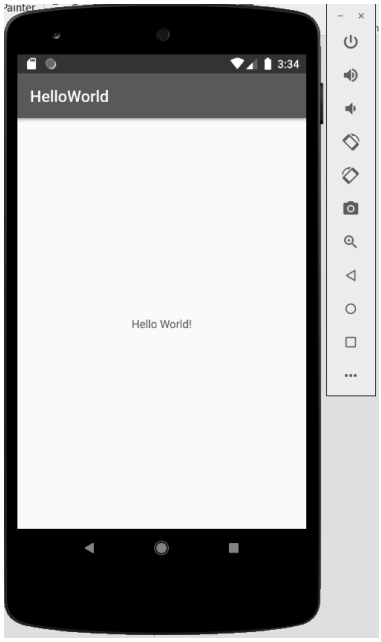
Slika 3.40, novi uređaj prikazan u AVD Manager ekranu

Kada je uređaj dodat, moguće je pokrenuti izvršavanje aplikacije na njemu klikom na dugme **Run app** (zelena strelica), kao što je prikazano na slici 3.41.



Slika 3.41, pokretanje aplikacije na dodatom virtuelnom uređaju

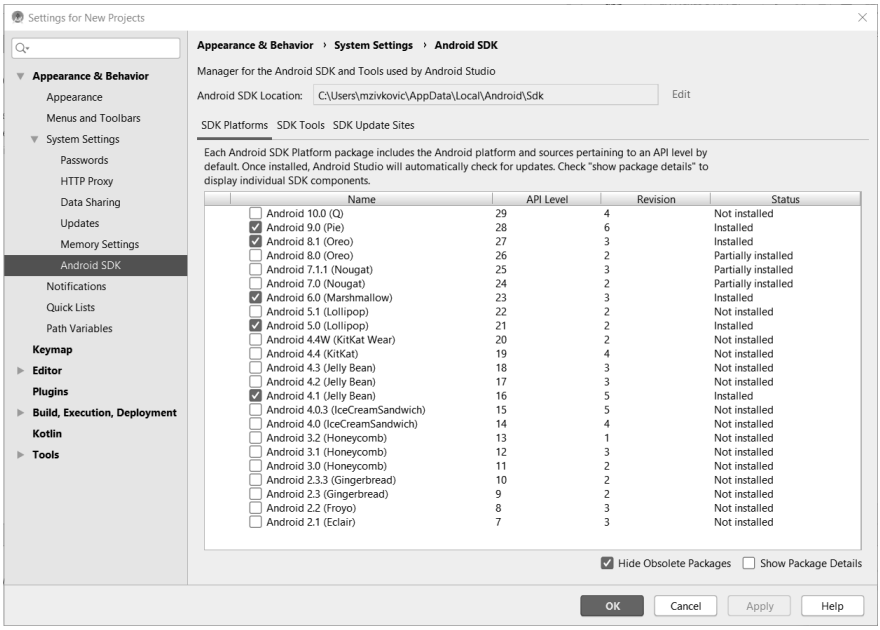
U zavisnosti od kompleksnosti virtuelnog uređaja, kao i od jačine računara na kojem se pokreće aplikacija, proces build-a aplikacije i pokretanja emulatora može da potraje nekoliko minuta. Zbog toga se preporučuje da kada se jednom podigne emulator, ne gasi dok se ne završi sa radom na aplikaciji, pošto će u tom slučaju svako sledeće pokretanje aplikacije ići znatno brže (pošto je emulator već podignut). „Hello World!“ aplikacija iz primera, pokrenuta na konfigurisanom virtuelnom uređaju, prikazana je na slici 3.42.



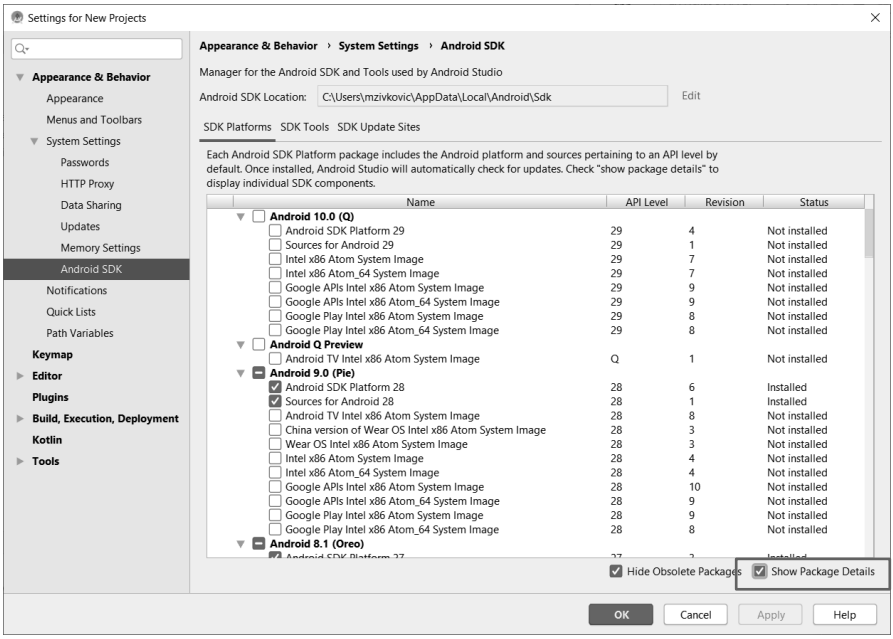
Slika 3.42, pokrenuta aplikacija na virtuelnom uređaju

3.4. SDK Manager

Android Studio nudi laku opciju preuzimanja novih SDK alata, platformi i drugih komponenti koje su potrebne za razvoj aplikacija, kao i njihovo lako održavanje i ažuriranje. U okviru sekcije **Tools,** odabirom stavke **SDK Manager** i taba **SDK Platforms**, moguće je videti listu svih dostupnih SDK, kao što je prikazano na slici 3.43. Neophodno je instalirati bar jednu verziju Android platforme u okruženju, kako bi bilo moguće kompajlirati aplikaciju. Dobra praksa je da se kao ciljna platforma koristi najnovija verzija SDK. Na taj način, aplikacija se kompajlira sa najnovijom verzijom biblioteka, čime se mogu koristiti najnovije funkcionalnosti koje mogu iskoristiti uređaji sa najnovijom verzijom Androida, a aplikacija se i dalje može izvršavati i na starijim verzijama Androida.

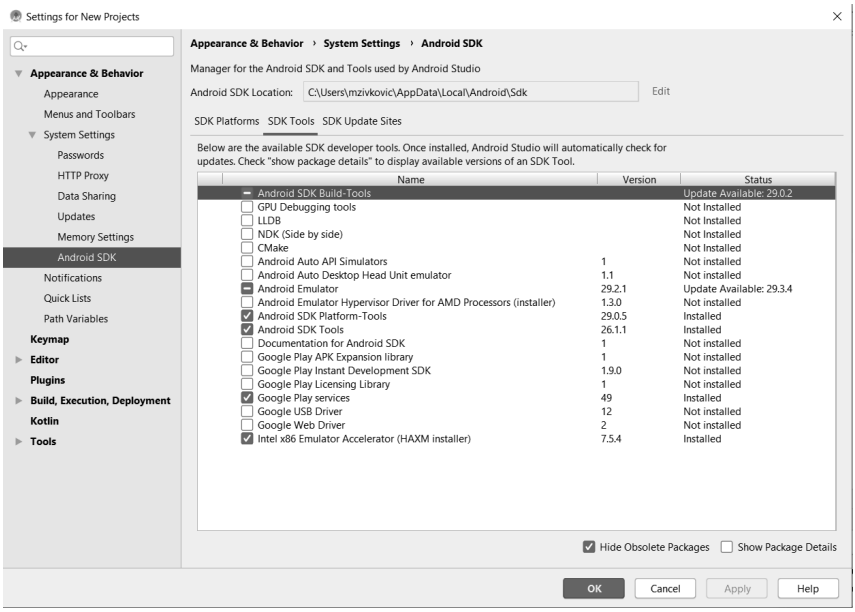


Slika 3.43, lista svih dostupnih SDK platformi



Slika 3.44, lista dostupnih SDK platformi sa detaljima za svaku pojedinačnu platformu

Za svaku dostupnu platformu moguće je videti njen status, odnosno da li je instalirana na računaru, kao i da li postoji dostupna novija verzija već instalirane platforme kroz opciju za ažuriranje. Takođe, moguće je videti detaljno sve komponente koje su raspoložive u okviru jednog SDK, klikom na dugme **Show Package Details**, kao što je prikazano na slici 3.44. Kroz ovaj prikaz moguće je instalirati samo one komponente koje su trenutno potrebne, a ne celu platformu. Željenu platformu ili komponentu je potrebno selektovati i zatim pritisnuti dugme **Apply**, čime će Android Studio započeti preuzimanje i instalaciju.



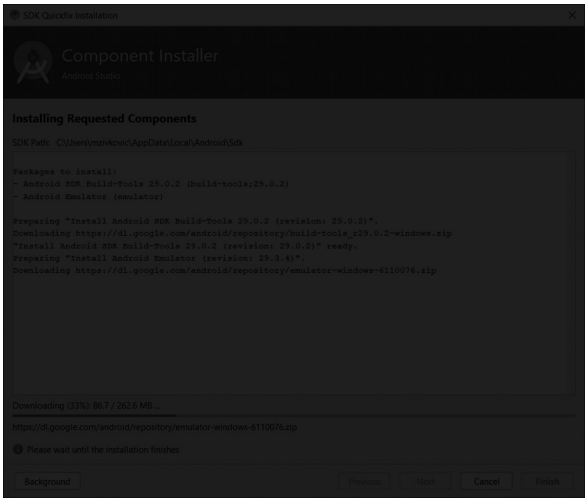
Slika 3.45, lista svih dostupnih SDK alata

U okviru taba **SDK Tools** moguće je videti listu svih dostupnih SDK razvojnih alata, kao što je prikazano na slici 3.45. Moguće je dodati nove alate, kao i ažurirati već instalirane ukoliko postoji nova verzija. Android Studio sam proverava dostupnost novih verzija, i označava sa crticom pakete koje je moguće ažurirati (na primer, na slici 3.45 vidi se da postoje nove verzije već instaliranih alata **Android SDK Build-Tools** i **Android Emulator**). Postoji veći broj dostupnih alata, manjeg ili većeg značaja, međutim, postoje alati koji su neophodni za normalan razvoj Android aplikacija koji bi trebalo da obavezno budu preuzeti, instalirani i dostupni u okruženju. Uz Android Emulator, u ove obavezne alate spadaju još:

- **Android SDK Build-Tools,** koji sadrži alate potrebne za build Android aplikacija.

- **Android SDK Platform-Tools**, koji sadrži veći broj važnih alata potrebnih Android platformi, poput adb.
- **Android SDK Tools,** sadrži ključne alate poput ProGuard (koji između ostalog služi za optimizaciju bajtkoda).

Ažuriranje i preuzimanje novih komponenti se prikazuje u obliku ekrana sa slike 3.46, a potrebno je napomenuti da je Android Studio nedostupan za vreme procesa ažuriranja.



Slika 3.46, proces ažuriranja komponenti

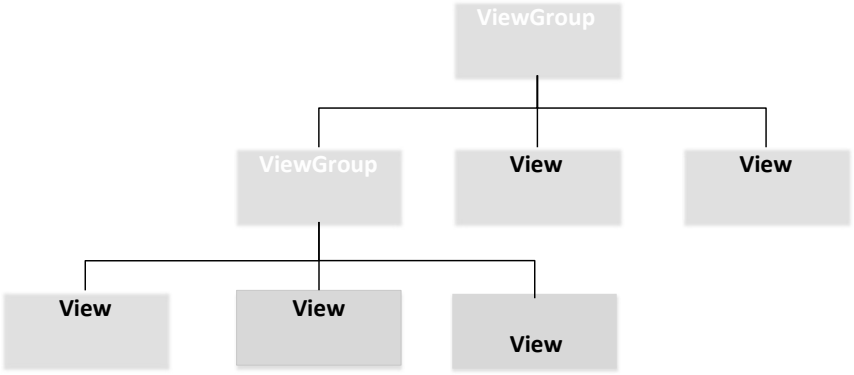
Kako bi se izvukao maksimum iz okruženja, neophodno je redovno ažurirati instalirane SDK i alate sa novim verzijama. Takođe, i samo okruženje treba redovno ažurirati, pošto će Android Studio obavestiti korisnika kada je dostupna nova verzija samog okruženja. Dobra praksa je da se nikada okruženje ne ažurira u toku jednog aktivnog projekta, pošto se dešava da izmene budu veće pa je neophodno utrošiti vreme u konfiguraciju i adaptaciju na novu verziju okruženja.

4. Grafički korisnički interfejs

Korisnički interfejs predstavlja sve što korisnik vidi na ekranu aplikacije i sa čime može da izvrši interakciju. To je prva i jedina tačka kontakta korisnika sa aplikacijom, kao i jedini deo aplikacije koji je vidljiv korisniku. Korisnički interfejs zbog toga mora biti funkcionalan i dopadljiv, zato što korisnici prilikom odabira između više sličnih aplikacija najčešće odluku donose samo na osnovu samog izgleda aplikacije (engl. *look and feel*). To je od naročitog značaja naročito u slučaju Android aplikacija, gde je konkurencija na tržištu (Google Play) ogromna.

Kada se programira korisnički interfejs za Android aplikaciju, koriste se gotove Androidove komponente korisničkog interfejsa, poput strukturiranih rasporeda komponenti i različitih elemenata i kontrola. Android nudi i složenije module poput dijaloga, notifikacija i različitih oblika menija. Osnovne Android komponente koje služe za kreiranje korisničkog interfejsa su klase View i ViewGroup. Sve komponente od kojih se sastoji korisnički interfejs su instance ove dve klase. U opštem slučaju, kao što je prikazano na slici 4.1, jedan korisnički interfejs Android aplikacije je kreiran kao hijerarhija rasporeda komponenti (*layout*) i vidžeta (*widgets*):

- Rasporedi komponenti su objekti koji pripadaju klasama izvedenim iz klase ViewGroup, sa ulogom da kontrolišu kako se njihova deca (View komponente ili drugi ViewGroup objekti) pozicioniraju na ekranu.
- Vidžeti su objekti klasa izvedenih iz klase View, iscrtavaju se na ekranu i sa njima korisnik aplikacije može da vrši interakciju (na primer dugmići ili polja za unos teksta).



Slika 4.1, primer strukture korisničkog interfejsa sa ViewGroup komponentama i pripadajućim View objektima

Kao što smo ranije naveli, filozofija razvoja Android aplikacija jeste da se u potpunosti razdvoje programski kod i resursi. Zbog toga, Android nudi XML rečnik za sve ViewGroup i View klase, tako da se korisnički interfejs definiše kroz XML fajl. View komponente se mogu definisati i u programskom kodu u Javi, u slučaju da je potrebno dinamički dodati komponente. Ipak, preporučeni način je da se komponente grafičkoj interfejsa definišu u XML fajlu. Komponentama koje su definisane u XML fajlu može se pristupiti iz koda, upotrebom R.java klase, o čemu će biti reči kasnije.

4.1. Klasa View

Klasa View predstavlja osnovni gradivni blok za komponente korisničkog interfejsa. Iz ove klase izvedene su sve ostale klase korisničkog interfejsa, tzv. vidžeti, koji se koriste za pravljenje interaktivnih komponenti korisničkog interfejsa, poput dugmića i polja za unos teksta. ViewGroup klasa, koja je bazna klasa za sve rasporede komponenti, takođe je izvedena iz klase View.

Svi View objekti jednog ekrana su poređani u jedno stablo. Ovo stablo treba da ima tačno jedan koreni element, koji je tipa ViewGroup. Kada je stablo kreirano i komponenta ubačena u njega, postoji određen broj tipičnih operacija koje se najčešće izvode.

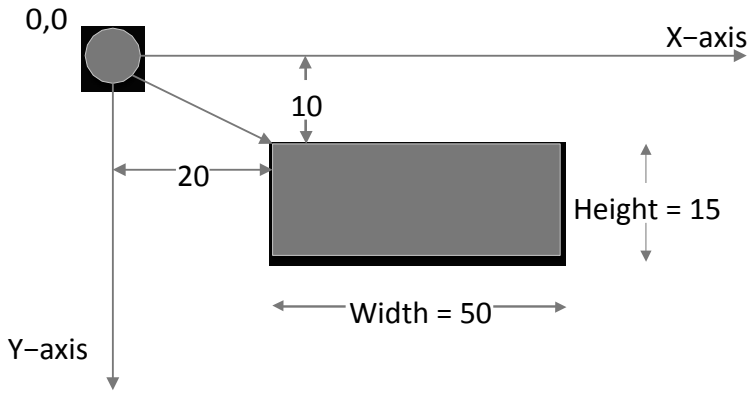
- Postavljanje atributa – poput postavljanja teksta u polje tipa TextView. Dostupne metode variraju u zavisnosti od konkretne izvedene klase.
- Postavljanje fokusa – pomeranje fokusa kao odgovor na neki korisnički unos. Fokus na određeno polje se može zahtevati pozivom metode requestFocus().
- Postavljanje osluškivača (engl. *listeners*) – View komponente dozvoljavaju postavljanje osluškivača, koji će biti obavešteni kada se desi nešto od interesa. Na primer, moguće je detektovati i obavestiti osluškivača kada komponenta dobije ili izgubi fokus. Izvedene klase imaju specijalizovane osluškivače, poput klase Button koja daje mogućnost da osluškivač bude obavešten kada se klikne na dugme.

View zauzima pravougaoni prostor na ekranu, i odgovoran je za crtanje komponente na ekranu, kao i za interakciju sa korisnikom, putem rukovanja događajima (engl. *event handling*). Svaki View ima svoju lokaciju na ekranu, koja je definisana pomoću para koordinata koje označavaju gornji levi ćošak (x, y), i dve dimenzije koje označavaju širinu i visinu (*width* i *height*), kao što je prikazano na slici 4.2. Svaka komponenta koja je tipa View ima raspoložive geter metode koje dohvataju odgovarajuće elemente pozicije komponente na ekranu, tako da getLeft() i getTop() služe da dohvate levi ćošak i vrate relativnu poziciju

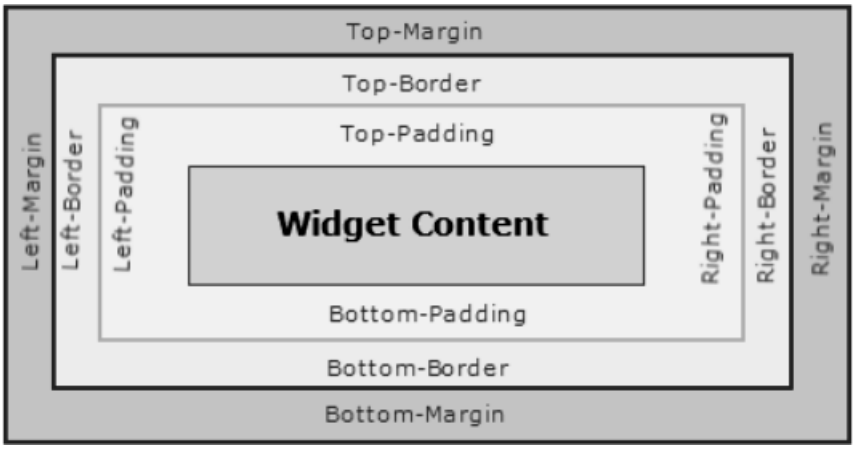
komponente u odnosu na roditeljsku komponentu. Analogno, getRight() i getBottom() služe za dohvatanje donjeg desnog ćoška. Metode getHeight() i getWidth() dohvataju iscrtanu visinu i širinu komponente (pri čemu važi getLeft()
+ getWidth() = getRight()).

Osim pozicije, bitni parametri za prikaz svake View komponente su njena veličina (*size*), *padding* i margine. Veličina komponente se izražava kroz visinu i širinu. Svaki View zapravo sadrži dva para vrednosti za visinu i širinu. Uz već pomenute metode getHeight() i getWidth() koje dohvataju stvarnu veličinu iscrtane komponente na ekranu, postoje još dve metode, pod nazivom getMeasuredHeight() i getMeasuredWidth(), koje vraćaju izmerenu visinu i širinu. Ove dve vrednosti zapravo pokazuju koliko komponenta želi da bude velika u svojoj roditeljskoj komponenti, i one mogu, ali ne moraju da budu jednake iscrtanim vrednostima.

Prilikom merenja svojih dimenzija, View mora da uračuna i padding. Padding se izražava u pikselima od leve, gornje, desne i donje ivice komponente, i služi da opiše rastojanje od granica komponente do samog sadržaja koji se nalazi u komponenti. Može se postaviti metodom setPadding(int, int, int, int), a dohvatiti get metodama getPaddingLeft(), getPaddingTop(), getPaddingRight() i getPaddingBottom(). Margine se sa druge strane odnose na prostor izvan granica komponente, odnosno definišu rastojanje od susednih komponenti (razlika između paddinga i margina prikazana je na slici 4.3). Margine se ne mogu podesiti na nivou pojedinačnih komponenti, već se postavljaju na nivou ViewGroup komponente koja služi za grupisanje komponenti.



Slika 4.2, definicija pozicije View komponente



Slika 4.3, padding i margine komponente

Svaki View može imati dodeljenu ID vrednost tipa integer. Ovi identifikatori se tipično dodeljuju u XML fajlu sa rasporedom komponenti postavljanjem atributa android:id, a služe za pronalaženje određene komponente u stablu. Ove vrednosti moraju biti jedinstvene. Na primer, u XML fajlu se može definisati dugme sa dodeljenim identifikatorom „mojeDugme“ na sledeći način:

```
<Button
     android:id="@+id/mojeDugme"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="@string/moj_tekst"/>
```

@+id naredba iz prethodnog fragmenta koda govori da je potrebno dodati novi identifikator u R.java klasu. Nakon toga se dugmetu iz odgovarajuće aktivnosti može pristupiti na sledeći način (preko R.java klase, koristeći dodeljeni identifikator):

```
Button mojeDugme = findViewById(R.id.mojeDugme);
```

4.2. Standardne komponente grafičkog interfejsa

Android nudi veći broj predefinisanih komponenti koje programer može da koristi. Sve ove komponente su izvedene iz bazne klase View, a programer može modifikovati njihov prikaz na ekranu pomoću atributa. U predefinisane komponente spadaju tipovi prikazani u tabeli 4-1.

Tip	Funkcionalnost	Klasa
Label	Tekstualno polje za prikaz teksta	TextView
Button	Dugme na koje korisnik može da klikne kako bi izvršio neku akciju	Button
Text field	Polje za unos teksta, uz podršku za automatsku dopunu teksta i različite tipove unosa (broj, mejl adresa, lozinka itd.)	EditText, AutoCompleteTextView
Checkbox	Polje za potvrdu koje korisnik može da čekira, pri čemu su polja međusobno nezavisna	CheckBox
Radio button	Slično kao Checkbox, uz razliku da su opcije zavisne jedna od druge	RadioGroup, RadioButton
Toggle	Dugme sa dva stanja (on/off)	ToggleButton
Spinner	Drop down lista iz koje korisnik može odabrati vrednost	Spinner
Pickers	Dijalog gde se vrednost bira upotrebom dugmeta gore/dole ili pokreta	DatePicker, TimePicker

Tabela 4-1, standardne komponente grafičkog interfejsa

Prikaz svake komponente moguće je kontrolisati skupom atributa. Na primer, svaki View ima atribute koji kontrolišu njegovu veličinu. Oni se moraju definisati u okviru XML fajla za svaki individualni View. Primer podešavanja veličine TextView komponente dat je sledećim isečkom koda iz XML fajla.

```
<TextView
 android:layout_width="150dp"
 android:layout_height="wrap_content" />
```

Atribut android:layout_width služi da odredi širinu komponente, i on se obavezno mora definisati. Analogno, android:layout_height služi za određivanje visine komponente. Oba atributa se mogu definisati u nekoliko različitih formi:

- wrap_content – prilagođava širinu i visinu komponente veličini samog sadržaja te komponente.
- match_parent – prilagođava širinu i visinu komponente veličini roditeljske komponente.
- pomoću dimenzije u dp (engl. *density independent pixels*) – na primer 150dp.
- referenciranjem vrednosti iz dimens.xml fajla (u obliku @dimen/dim)

Ove vrednosti se mogu promeniti za vreme izvršavanja iz Java koda na nekoliko načina, na primer:

```
// Promena širine i visine
int novaSirina = 50; //u običnim pikselima
int novaVisina = 50;
view.setLayoutParams(new LayoutParams(novaSirina,
novaVisina));
// zahtevamo invalidaciju view kako bi se ponovo
iscrtala view.requestLayout();
```

Još jedan bitan atribut svake komponente je gravity, koji služi da se definiše na koju stranu treba da „gravitira“ sadržaj u okviru komponente. Dolazi u dva oblika:

- gravity – definiše smer u kome se sadržaj View komponente poravnava (veoma slično kao CSS text-align).
- layout_gravity – definiše smer u kome View gravitira unutar svoje roditeljske komponente (poput CSS float).

Dozvoljene vrednosti koje se mogu koristiti su: left, right, top, bottom, center, itd. Moguće je specificirati i više vrednosti, pri čemu se mora koristiti vertikalna crta za razdvajanje, na primer: android:gravity=”top|right”. Primer primene ovih atributa na komponentu tipa TextView dat je sledećim XML kodom:

```
<TextView
  android:gravity="left"
  android:layout_gravity="right"
  android:layout_width="150dp"
  android:layout_height="wrap_content"
  android:textSize="10sp"
  android:text="@string/hello_world" />
```

Klasa View ima veliki broj atributa koji se mogu primeniti na komponentu, kako bi se kontrolisali različiti aspekti komponente. Neki od najčešće korišćenih atributa su prikazani u tabeli 4-2. Ova lista nije konačna, već samo prikazuje pregled često korišćenih atributa. Uz svaki atribut, dat je i kratak opis, kao i primer vrednosti koja mu se može dodeliti.

Atribut	Opis	Primer vrednosti
android:background	Postavljanje pozadine za View	#ffffff
android:onClick	Metoda koja se poziva nakon klika na komponentu	onButtonClicked
android:visibility	Kontroliše prikaz komponente	invisible
android:hint	Hint tekst koji se prikazuje kada je komponenta prazna	@string/hint
android:text	Tekst koji se prikazuje u komponenti	@string/poruka
android:textColor	Boja teksta	#000000
android:textSize	Veličina teksta	21sp
android:textStyle	Stil formatiranja teksta	bold

Tabela 4-2, najčešće korišćeni atributi sa primerima vrednosti

Primer postavljanja više atributa za komponentu tipa TextView dat je sledećim isečkom XML koda. Ovim primerom se tekst koji se prikazuje u okviru komponente referencira iz strings.xml fajla, prikazuje se u beloj boji, na crnoj pozadini.

```
<TextView
   android:layout_width="150dp"
   android:layout_height="wrap_content"
   android:text="@string/poruka"
   android:background="#000"
   android:textColor="#fff"
/>
```

U nastavku su dati opisi osnovnih komponenti interfejsa opisanih u tabeli 4-1, sa tipičnim primerima upotrebe.

4.2.1. TextView

TextView je komponenta korisničkog interfejsa koja služi za prikaz teksta korisniku. Podrazumevano ponašanje TextView komponente jeste da tekst nije promenljiv, odnosno ponaša se kao labela na ekranu korisničkog interfejsa. Ukoliko je potrebno omogućiti korisniku da menja ili unosi tekst, koristi se EditText, koji je opisan nešto kasnije. Ključni atributi komponente tipa TextView (uz već navedene u prethodnom poglavlju) su:

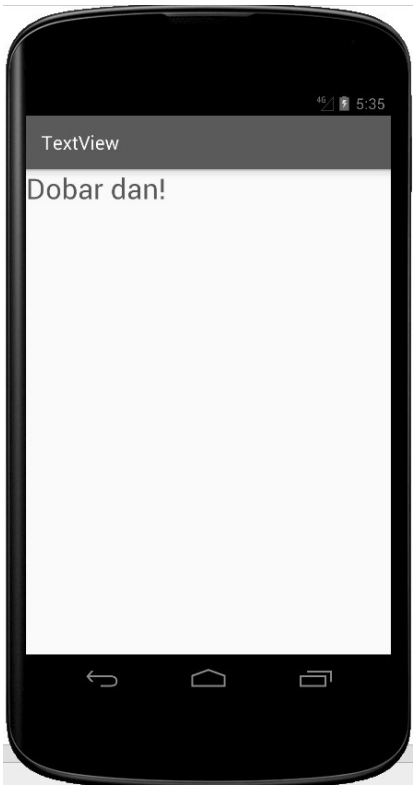
- android:id – opciono polje koje omogućava pristup ovoj komponenti iz Java koda, preko R.java klase, kao i referenciranje na ovu komponentu iz drugih komponenti u rasporedu.
- android:text – definiše se tekst koji će biti prikazan kao sadržaj komponente. Moguće je da bude direktno data („hard-coded“) fiksna vrednost, a može i da bude referenca na String resurs koji je definisan u strings.xml (@string naredba).

Tipičan primer upotrebe komponente tipa TextView dat je u sledećem primeru XML koda:

```
<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/labelaText"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
```

```
        android:textSize="30dp"
        android:text="Dobar dan!" />
</LinearLayout>
```

Ovako kreirana TextView komponenta će na ekranu biti iscrtana na način prikazan na slici 4.4.



Slika 4.4, prikaz tipične upotrebe TextView komponente

Komponenti se može pristupiti i iz programskog koda. Moguće je izmeniti sadržaj komponente definisane u XML fajlu tako što se željena komponenta referencira iz koda pomoću njenog ID uz pomoć R.java klase. Primer modifikovanja sadržaja dat je kroz jednostavnu aktivnost koja dohvata TextView polje i menja mu sadržaj novim Stringom koji se nalazi definisan u strings.xml fajlu. Primer podrazumeva da je sadržaj fajla strings.xml, koji se nalazi u res/values folderu, dat sa:

```
<resources>
    <string name="app_name">Korisnicki interfejs</string>
    <string name="nova_poruka">Dobrodosli na RMA!</string>
</resources>
```

U tom slučaju, kod MainActivity.java klase dat je sa:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView helloTextView =
findViewById(R.id.labelaText);
        helloTextView.setText(R.string.nova_poruka);
    }
}
```

Dimenzija teksta u prethodnom primeru zadata je atributom android:textSize. Ona može biti definisana na više načina, odnosno u više različitih mernih jedinica. Moguće jedinice koje se mogu koristiti su:

- px – veličina slova zadata u pikselima.
- in – veličina slova zadata u inčima.
- mm – veličina slova zadata u milimetrima.
- sp – veličina slova zadata u skaliranim pikselima baziranim na veličini slova koju je korisnik odabrao kao *preferred font size* u celom Android sistemu.
- dp – *density-independent pixels*.

Jedinica sa oznakom dp je apstraktna merna jedinica koja zavisi od fizičke gustine tačaka ekrana. Jedinica je zadata relativno u odnosu na 160 dpi (*dots per inch*) ekran, na kome važi da je 1dp ekvivalentan 1px. Kada se aplikacija izvršava na ekranu sa većom gustinom, broj piksela koji se koristi za crtanje 1dp se skalira naviše odgovarajućim faktorom koji zavisi od dpi ekrana. Analogno, ukoliko se aplikacija izvršava na ekranu sa manjom gustinom, broj piksela koji se koristi za crtanje 1dp se skalira naniže. U opštem slučaju, u vreme izvršavanja, dimenzija zadata u dp se preračunava po formuli px = dp * (dpi / 160). Ovo je preporučeni način predstavljanja dimenzija komponenti, pošto će se ispravno iscrtavati na ekranima sa različitim gustinama tačaka.

4.2.2. EditText

EditText je komponenta korisničkog interfejsa koja služi za unos ili modifikaciju teksta. Direktno je izvedena iz klase TextView, a služi kao bazna klasa za klasu AutoCompleteTextView koja je izvedena iz nje. Kada se dodirne ovo polje, na njega se postavlja kursor i podiže virtuelna tastatura u kojoj korisnik može da

unese tekst, a dozvoljeno je da radi i copy/paste. Ključni atributi komponente EditText su:

- android:inputType – ovaj atribut konfiguriše tip virtuelne tastature koja će se prikazati, dozvoljene karaktere, kao i izgled polja. Na primer, ukoliko polje služi za unos neke tajne informacije, poput lozinke, ovaj atribut se može postaviti na vrednost textPassword, čime se otvara standardna virtuelna tastatura, a uneti karakteri se ne prikazuju na ekranu već se umesto njih prikazuju tačkice. Odabir number vrednosti za ovaj atribut će ograničiti unos na brojeve, a prikazaće se numerička tastatura. Druge moguće vrednosti uključuju textEmailAddress, phone, numericPassword itd. pri čemu svaka od opcija ima predefinisanu virtuelnu tastaturu.
- android:hint – prikaz pomoćnog teksta koji korisniku treba da sugeriše šta treba uneti u polje.

Tipičan primer upotrebe EditText komponente dat je u sledećem primeru XML koda:

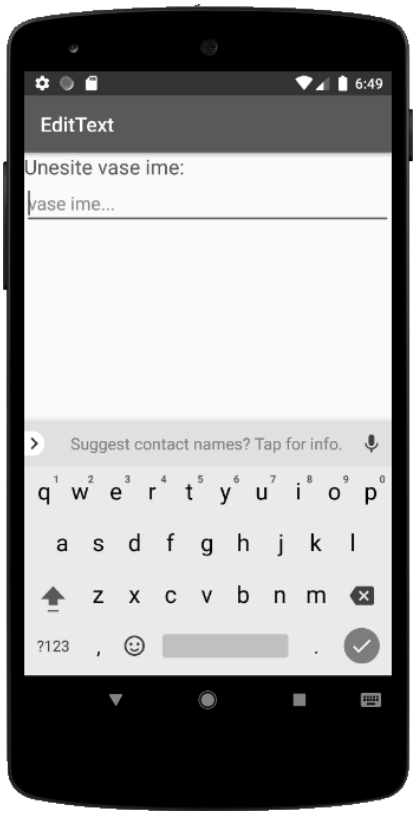
```
<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/labelaPoruka"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:textSize="20dp"
        android:text="Unesite vase ime:" />
    <EditText
        android:id="@+id/inputIme"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:hint="vase ime..."
        android:inputType="text"/>
</LinearLayout>
```

Komponenti se može pristupiti i iz programskog koda. Deo Java koda, koji bi se nalazio negde u MainActivity.java na mestu gde je potrebno dohvatiti unetu vrednost, i koji bi preko R.java klase pristupio sadržaju koji je korisnik uneo u polje i sačuvao ga u promenljivu tipa String, dat je sa:

```
EditText inputIme = findViewById(R.id.inputIme);
String unetoIme = inputIme.getText().toString();
```

Nakon pokretanja aplikacije, ovako definisana komponenta biće iscrtana kao na slici 4.5. Dodir na polje za unos će otvoriti standardnu virtuelnu tastaturu, pošto je inputType polja definisan kao text. Drugačije definisana vrednost inputType prikazaće i drugačiju tastaturu, kao što je prikazano na slici 4.6. Primer XML koda koji bi se koristio za definisanje EditText polja za unos telefona (inputType phone), dat je u nastavku:

```
<EditText
    android:id="@+id/inputPhone"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/phone_hint"
    android:inputType="phone" />
```



Slika 4.5, prikaz tipične upotrebe EditText komponente

Slika 4.6, različiti tipovi virtuelne tastature za različite vrednosti inputType atributa

4.2.3. Button

Button je komponenta korisničkog interfejsa koja prikazuje dugme. Dugme je objekat koji korisnik može da dodirne ili klikne na njega kako bi se izvršila neka akcija. Bitni atributi klase Button su:

- android:text – definiše tekst koji je ispisan na dugmetu.
- android:onClick – opciono, definiše se ime metode iz odgovarajuće aktivnosti čije će se izvršavanje okinuti kada se dugme klikne.

Tipičan primer definisanja dugmeta dat je u sledećem XML kodu.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/labelaPoruka"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="Pritisnite dugme" />
    <Button
        android:id="@+id/button"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="Pritisni me!"/>
</LinearLayout>
```

Ovako definisano dugme će biti uredno prikazano na ekranu, međutim, ništa se neće dogoditi kada korisnik klikne na njega. Kako bi se nešto zaista desilo kada korisnik klikne na dugme, programer mora napisati metodu koja će predstavljati reakciju na klik. U ovom kontekstu, klik na dugme prestavlja događaj koji je korisnik generisao (engl. *event*). Kada se desi klik, objekat tipa Button će dobiti on-click događaj. Sledeća stvar koju programer treba da napiše jeste obrada događaja, koja je zapravo komad koda koji se piše unutar klase odgovarajuće aktivnosti. U opštem slučaju, to se može postići na dva načina.

Prvi način za obradu događaja jeste kroz Java kod odgovarajuće aktivnosti, i ovo je preferirani način obrade događaja. U tom slučaju, XML kod ostaje isti, dok se kompletna obrada događaja piše u kodu. Najpre je neophodno kreirati instancu klase View.OnClickListener, koja implementira istoimeni interfejs, koji je definisan u okviru klase View. U ovom primeru, koristi se objekat anonimne klase u pozivu metode setOnClickListener nad objektom tipa Button, čija je referenca dobijena preko R.java klase pozivom metode findViewById(). Pošto je u pitanju anonimna klasa koja implementira interfejs, neophodno je dati implementaciju nedostajućih metoda. Interfejs View.OnClickListener ima samo jednu deklarisanu apstraktnu metodu, onClick(), koju je potrebno nadjačati i implementirati. To je metoda koja će biti pozvana kada se on-click događaj zaista desi, a ovakve metode su poznate kao callback metode. Unutar ove metode se samo poziva Toast preko kojeg će se korisniku prikazati kratka poruka na ekranu.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = findViewById(R.id.button);
        button.setOnClickListener(new
View.OnClickListener() {
            public void onClick(View v) {
                // Kod koji se izvrsava kada se klikne na
dugme
                Toast.makeText(getApplicationContext(),
"Klik!", Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Drugi način jeste da se obrada događaja definiše kroz XML fajl. U tom slučaju, prvo je neophodno malo izmeniti sam XML fajl. Potrebno je dodati atribut android:onClick, čija je vrednost zapravo ime metode koja treba da se pozove kada korisnik klikne na dugme. Izmenjeni XML fajl dat je u nastavku:

```
<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/labelaPoruka"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:textSize="20dp"
        android:text="Pritisnite dugme" />
    <Button
        android:id="@+id/button"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:onClick="prikaziPoruku"
        android:text="Pritisni me!"/>
</LinearLayout>
```

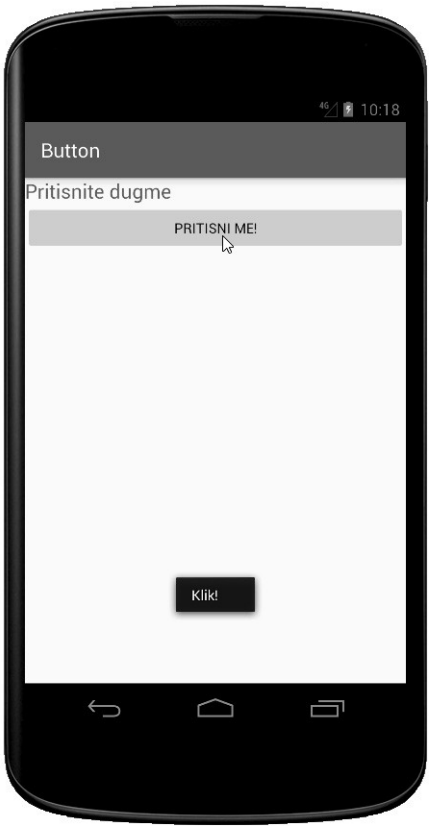
Na ovaj način definisano je kroz XML da kada se desi događaj klika na dugme, treba pozvati metodu čije je ime dato kao vrednost android:onClick atributa, odnosno u ovom slučaju metodu prikaziPoruku(). Ova metoda mora da se nalazi u aktivnosti kojoj odgovara ovaj XML fajl, i mora da ispunjava tri uslova:

- public modifikator pristupa – ukoliko bi bilo private, to znači da se metodi može pristupiti samo iz klase u kojoj je definisana, a ne iz pridruženog XML fajla.
- parametar tipa View – metoda mora imati ovaj parametar, jer će kroz njega biti prosleđen View koji je generisao događaj.
- povratni tip metode mora biti void.

Uz ova navedena dva uslova, java kod aktivnosti sada izgleda ovako:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void prikaziPoruku(View v){
        Toast.makeText(getApp licationContext(),
"Klik!", Toast.LENGTH_LONG).show();
    }
}
```

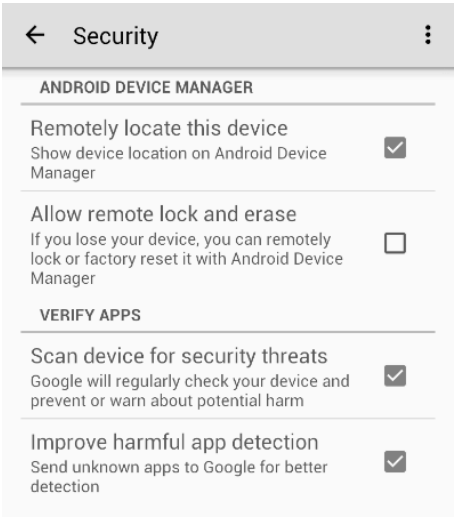
Kod metode prikaziPoruku je identičan kao kod onClick metode koji je upotrebljen u prvom opisanom pristupu. Svejedno koji se pristup koristi, rezultat je isti – nakon klika na dugme prikazaće se mali Toast sa tekstom „Klik!“ na ekranu, kao što je prikazano na slici 4.7. Kao što je već napomenuto, iako se na oba načina može postići isti rezultat, preferirani način je da se to uradi na prvi način, kroz Java kod.



Slika 4.7, tipičan primer upotrebe dugmeta sa jednostavnom obradom događaja

4.2.4. CheckBox

Komponente tipa CheckBox omogućavaju korisniku da selektuje jednu ili više opcija iz skupa opcija. Tipično se ove komponenteređaju u vertikalnoj listi – na taj način ih sam Android koristi kroz sistem, kao na slici 4.8. Svaki pojedinačni CheckBox može biti u jednom od dva stanja – čekiran i nečekiran, nezavisno od stanja drugih CheckBox komponenti.



Slika 4.8, komponente CheckBox

Za kreiranje svake pojedinačne opcije, kreira se po jedan CheckBox. Pošto je moguće da korisnik odabere i više opcija u isto vreme, svaka opcija se mora pratiti pojedinačno, odnosno mora se registrovati osluškivač za svaku pojedinačnu opciju. Kada korisnik selektuje jednu opciju, taj CheckBox objekat dobija on-click događaj. Slično kao u primeru za Button, rukovanje događajem se može implementirati na dva načina – kroz XML i kroz Java kod.

Ukoliko se rukovanje događajem implementira kroz XML, potrebno je svakom pojedinačnom CheckBox elementu u XML fajlu dodati android:onClick atribut, čija je vrednost ime metode koju želimo da pozovemo kao odgovor na događaj. Naravno, aktivnost koja odgovara XML fajlu mora implementirati taj metod. XML kod i Java kod odgovarajuće aktivnosti dati su u nastavku.

```
<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/labelaPoruka"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:textSize="20dp"
        android:text="Odaberite dodatke za kafu" />
    <CheckBox android:id="@+id/checkbox_mleko"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:text="Mleko"
    android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkbox_secer"
        android:layout_width="wrap_content"
        android:layout_he ight="wrap_content"
        android:text="Secer"
        android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

U slučaju klika na bilo koji od dve opcije, poziva se metoda onCheckboxClicked(), koja mora biti implementirana u odgovarajućoj aktivnosti. U samoj metodi neophodno je znati na koju konkretnu opciju je korisnik kliknuo. To se postiže proverom Id komponente v tipa View, koja je generisala događaj i koja je prosleđena kao parametar u metodi. Id se dobija pozivom metode v.getId(), i zatim kroz switch naredbu poredi sa Id svake pojedinačne opcije. Kod odgovarajuće aktivnosti dat je u nastavku.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onCheckboxClicked(View v){
        // Da li je view sada cekiran?
        boolean checked = ((CheckBox) v).isChecked();
        // Proveri koji checkbox je upravo kliknut
        switch(v.getId()) {
            case R.id.checkbox_mleko:
                if (checked){
                    //dodaj mleko u kafu
                } else{
                    //nemoj stavljati mleko
                }
                break;
            case R.id.checkbox_secer:
                if (checked){
                    //dodaj secer u kafu
                } else {
                    //nemoj stavljati secer
                }
                break;
            // TODO: druge opcije
        }
    }
}
```

Ukoliko se koristi drugi pristup, gde se rukovanje događajem kompletno implementira u kodu, XML fajl i kod odgovarajuće aktivnosti dati su u nastavku. XML fajl je skoro identičan, osim uklonjenih android:onClick atributa.

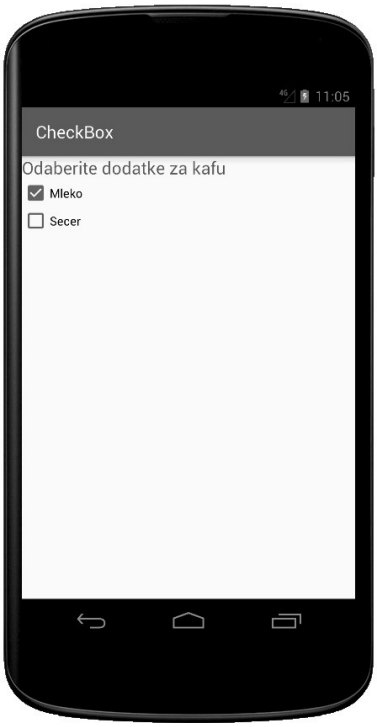
```
<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/labelaPoruka"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:textSize="20dp"/>
    <CheckBox android:id="@+id/checkbox_mleko"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mleko">
    <CheckBox android:id="@+id/checkbox_secer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Secer"/>
</LinearLayout>
```

Odgovarajuća aktivnost je implementirana nešto drugačije. Programeri vrlo često žele da izbegnu rad sa anonimnim klasama, pa se često sama klasa aktivnosti proglasi za osluškivač. To se postiže tako što se u zaglavlju klase proglasi da klasa implementira View.OnClick interfejs. Na taj način, prilikom postavljanja osluškivača na odgovarajuću komponentu, kao parametar metode setOnClickListener() aktivnost može proslediti samu sebe, prosleđivanjem vrednosti this. To je moguće pošto setOnClickListener() metoda očekuje objekat neke klase koja implementira interfejs View.OnClick, a naša aktivnost jeste takav objekat. Kako bi se ispoštovala implementacija interfejsa, u samoj klasi aktivnosti potrebno je implementirati onClick() metodu definisanu u interfejsu View.OnClick, tipa void i sa jednim parametrom tipa View – to je komponenta koja je generisala događaj.

```
public class MainActivity extends AppCompatActivity
implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        CheckBox checkBoxMleko =
```

```
findViewById(R.id.checkbox_mleko);
        checkBoxMleko.setOnClickListener(this);
        CheckBox checkBoxSecer =
findViewById(R.id.checkbox_secer);
        checkBoxSecer.setOnClickListener(this);
    }

    public void onClick(View v){
        // Da li je view sada cekiran?
        boolean checked = ((CheckBox) v).isChecked();
        // Proveri koji checkbox je upravo kliknut
        switch(v.getId()) {
            case R.id.checkbox_mleko:
                if (checked){
                    //dodaj mleko u kafu
                } else{
                    //nemoj stavljati mleko
                }
                break;
            case R.id.checkbox_secer:
                if (checked){
                    //dodaj secer u kafu
                } else {
                    //nemoj stavljati secer
                }
                break;
            // TODO: druge opcije
        }
    }
}
```

U oba pristupa, ponašanje aplikacije je identično. Prikaz aplikacije dat je na slici 4.8.



Slika 4.9, tipičan primer upotrebe CheckBox komponenti

4.2.5. RadioButton

Komponente tipa RadioButton su komponente sa dva stanja (čekiran ili nečekiran), i veoma su slične CheckBox komponentama, uz bitnu razliku da RadioButton komponente rade u grupi. Unutar jedne grupe RadioButton komponenti samo jedna može biti odabrana u jednom trenutku. Klik na bilo koji drugi RadioButton u okviru iste grupe odčekiraće sve ostale RadioButton komponente te grupe. Grupa se definiše RadioGroup objektom.

RadioButton komponente omogućavaju korisniku da odabere jednu opciju iz skupa opcija. Koriste se u slučaju da je potrebno da opcije budu međusobno isključive, a želimo da sve ponuđene opcije budu prikazane jedna pored druge. Ukoliko nije potrebno da budu prikazane jedna pored druge već jedna ispod druge, preporučuje se da se koristi Spinner. Kreiraju se vrlo slično kao CheckBox komponente, ali se moraju grupisati zajedno unutar grupe – RadioGroup elementa. Na taj način se obezbeđuje da samo jedna komponenta iz jedne grupe može biti odabrana u jednom trenutku. RadioGroup je izvedena iz LinearLayout, i podrazumevano ima vertikalnu orijentacijuređanja komponenti. Tipičan primer upotrebe dat je u nastavku, sledećim XML fajlom:

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <RadioButton android:id="@+id/radio_zvezda"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Zvezda"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_partizan"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Partizan"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

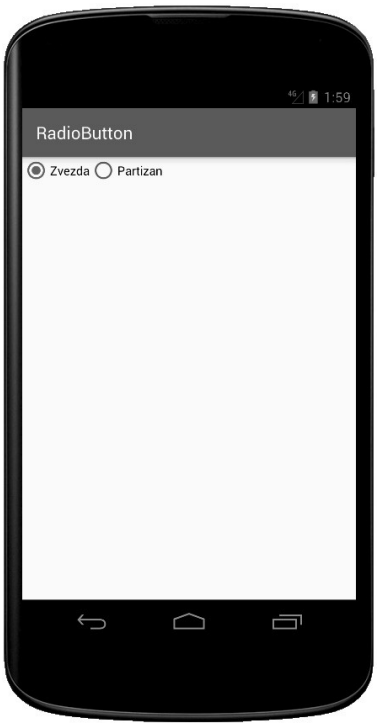
U ovom primeru se od korisnika traži da iskaže svoje navijačko opredeljenje, a ponuđene opcije su isključive. Bilo koja RadioButton komponenta da se odabere, pozvaće se metoda onRadioButtonClicked, koja se mora implementirati u odgovarajućoj aktivnosti. I ovde se rukovanje događajem može implementirati na dva načina, veoma slično kao i kod CheckBox komponenti. Zbog izrazite sličnosti, prikazana je implementacija rukovanja događajem kroz XML, dok se implementacija kompletnog rukovanja kroz Java kod realizuje analogno i preporučuje se čitaocu kao vežba.

Implementacija aktivnosti koja odgovara ovom XML fajlu data je u nastavku. Slično kao kod CheckBox komponenti, u metodi onRadioButtonClicked se prvo proverava koja komponenta v tipa View je generisala događaj, dohvatanjem vrednosti v.getId() i upotrebom naredbe switch da se ta vrednost uporedi sa Id svakog RadioButton elementa ponaosob. Izgled aplikacije nakon pokretanja prikazan je na slici 4.10.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onRadioButtonClicked(View view) {
        // Da li je dugme sada cekirano?
        boolean checked = ((RadioButton)
        view).isChecked();

        // Proveri koje radio dugme je kliknuto
```

```
        switch(view.getId()) {
```

```
                case R.id.radio_zvezda:
                    if (checked)
                        // Zvezda je najbolja!
                        break;
                case R.id.radio_partizan:
                    if (checked)
                        // Partizan je najbolji
                        break;
            }
        }
}
```



Slika 4.10, tipičan primer upotrebe RadioButton komponente

4.2.6. Spinner

Spineri omogućavaju brz odabir jedne vrednosti iz zadatog skupa vrednosti. Spiner pokazuje trenutno odabranu vrednost kao podrazumevanu vrednost u prikazu. Dodir na Spiner prikazuje klasičan padajući meni sa svim dostupnim vrednostima, od kojih korisnik može odabrati jednu.

Spiner se jednostavno dodaje u XML rasporedu ubacivanjem Spinner elementa. Na primer, u sledećem XML fajlu dodat je spiner koji će kao vrednosti sadržavati sve planete Sunčevog sistema:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <TextView
        android:id="@+id/labelPlanete"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Planete" />
    <Spinner
        android:id="@+id/spinner"
        android:layout_w idth="match_parent
        "
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
</LinearLayout>
```

Kako bi se spiner popunio sa listom opcija, u aktivnosti se mora specificirati SpinnerAdapter. Opcije kojima se popunjava spiner mogu doći iz bilo kog izvora, ali se moraju proslediti kroz jedan SpinnerAdapter, na primer kroz ArrayAdapter ukoliko su opcije dostupne u nizu vrednosti, ili kroz CursorAdapter ukoliko su opcije dostupne kao rezultat upita u bazu. Na primer, ukoliko su opcije za spiner predefinisane, one se mogu definisati u obliku niza Stringova u fajlu strings.xml. Primer sadržaja strings.xml fajla dat je sledećim isečkom XML koda:

```
<resources>
    <string name="app_name">Spinner</string>
    <string-array name="niz_planeta">
        <item>Merkur</item>
        <item>Venera</item>
        <item>Zemlja</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uran</item>
        <item>Neptun</item>
    </string-array>
</resources>
```

Kod aktivnosti koja odgovara datom XML fajlu prikazan je u nastavku. Metoda createFromResource() omogućava kreiranje ArrayAdapter objekta iz niza

Stringova. Treći argument ove metode predstavlja layout resurs koji definiše

kako će se odabrana opcija prikazati u spineru – simple_spinner_item je podrazumevani layout koji je obezbeđen od strane Android platforme, i koji treba da se koristi uvek (osim ukoliko programer ne želi da samostalno definiše layout za prikaz spinera). Nakon toga, nad objektom klase adapter poziva se metoda setDropDownViewResource(int), koja specificira layout koji će adapter koristiti da prikaže listu spiner opcija (simple_spinner_dropdown_item je takođe standardna opcija koju obezbeđuje Android). Nakon toga se poziva metoda setAdapter() nad objektom tipa Spinner kako bi se kreirani adapter primenio. Izgled samog spinera prikazan je na slici 4.11, gde se vidi prikaz odabrane komponente, kao i prikaz liste komponenti nakon dodira na komponentu spiner.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Spinner spinner = (Spinner)
findViewById(R.id.spinner);
        // Kreira se ArrayAdapter koji koristi
niz stringova I podrazumevani spinner layout
        ArrayAdapter<CharSequence> adapter =
ArrayAdapter.createFromResource(this,
                R.array.niz_planeta,
android.R.layout.simple_spinner_item);
        // Specificira se layout koji se koristi kada
se pojavi lista opcija
adapter.setDropDownViewResource(android.R.layout.
simple_spinner_dropdown_item);
        // primenjuje se adapter na spinneru
        spinner.setAdapter(adapter);
    }

}
```

Dodatno, potrebno je navesti da kada korisnik odabere neku od opcija iz padajuće liste, Spinner objekat dobija on-item-selected događaj. Moguće je dodati u aktivnost kod kojim se rukuje ovim tipom događaja, implementacijom interfejsa osluškivača AdapterView.OnItemSelectedListener i dodavanjem odgovarajuće onItemSelected() callback metode. Potrebno je dodati i implementaciju onNothingSelected() metode, koju zahteva implementirani interfejs. Primer upotrebe ovog osluškivača je dat sa:

```
public class SpinnerActivity extends Activity implements
OnItemSelectedListener {
    ...
    public void onItemSelected(AdapterView<?> parent, View
view,
            int pos, long id) {
//  Opcija je odabrana, i moze se dohvatiti upotrebom
// parent.getItemAtPosition(pos)
    }

    public void onNothingSelected(AdapterView<?> parent) {
        // Another interface callback
    }
}
```

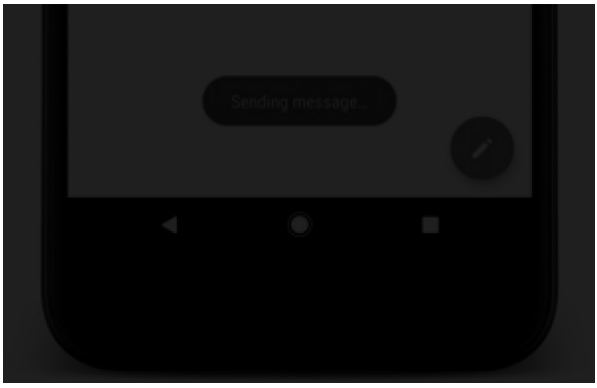
Postavljanje ovog osluškivača na objekat tipa Spinner se radi na sledeći način:

```
Spinner spinner = findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(this);
```

Slika 4.11, tipičan primer upotrebe Spinner komponente

4.2.7. Toast

Toast služi za prikaz jednostavnih poruka korisniku aplikacije putem malog pop- up prozora. Obično se koristi da se korisniku da povratna informacija o nekoj izvršenoj akciji. Na primer, ukoliko se u mejl aplikaciji klikne na Send, obično se prikaže kratka „Sending message…“ Toast poruka, kao što je prikazano na slici 4.12. Na Toast poruku nije moguće kliknuti. Ona prilikom prikaza na ekranu zauzima samo prostor potreban za ispis poruke, pri čemu trenutna aktivnost ostaje vidljiva i interaktivna. Toast automatski nestaje nakon



određenog vremena.

Slika 4.12, prikaz kratke Toast poruke u mail aplikaciji
(slika preuzeta sa: https://developer.android.com/guide/topics/ui/notifiers/toasts)

Kreiranje Toast poruke je veoma jednostavno. Prvo, potrebno je napraviti instancu Toast klase pozivom makeText() metode. Ova metoda prihvata tri parametra: kontekst aplikacije, tekstualnu poruku i dužinu prikaza Toast poruke na ekranu, i kao rezultat vraća ispravno inicijalizovani Toast objekat. Prikaz Toast poruke na ekranu postiže se pozivom metode show() nad Toast objektom. Tipičan primer upotrebe Toast poruke dat je sledećim Java kodom.

```
Context context = getApplicationContext();
CharSequence text = "Zdravo, ovo je Toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

Obično se koristi još kraća i jednostavnija sintaksa. Moguće je koristiti ulančavanje metoda i izbeći potrebu za Toast promenljivom, na sledeći način:

```
Toast.makeText(getApplicationContext(), text,
Toast.LENGTH_LONG).show();
```

Navedeni primer je standardan način korišćenja Toast poruke, dovoljan za sve primene u aplikacijama. Moguće je uticati na dužinu prikaza sa dve opcije: Toast.LENGTH_SHORT za kratak prikaz, i Toast.LENGTH_LONG za duži prikaz. Retko kad je potrebno nešto dodatno uraditi. Na primer, može se uticati na poziciju gde će se Toast prikazati upotrebom setGravity() metode. Standardna Toast notifikacija se prikazuje pri dnu ekrana, centrirana horizontalno. Metoda setGravity() prihvata tri parametra: Gravity konstantu, pomeraj po x osi i pomeraj po y osi. Na pimer, ukoliko bismo odlučili da Toast poruka treba da se prikaže u gornjem levom uglu ekrana, potrebno bi bilo uraditi sledeće:

```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

Ukoliko bismo želeli da gurnemo poziciju prikaza poruke u desnu stranu, trebalo bi povećati drugi parametar. Ukoliko želimo da pomerimo poruku na dole, trebalo bi povećati treći parametar).

Primer jednostavne aplikacije koja prikazuje Toast poruku svaki put kad se klikne na dugme je dat u nastavku.

XML fajl:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView
        android:id="@+id/labelDemo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:text="Toast primer" />
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Prikazi Toast" />
</LinearLayout>
```

Java kod odgovarajuće aktivnosti, sa implementacijom osluškivača za on-click događaj je dat u nastavku. Osluškivač je realizovan tako što sama klasa aktivnosti implementira odgovarajući View.OnClickListener, pa je moguće prilikom postavljanja osluškivača na objekat tipa Button proslediti metodi setOnClickListener() parametar this u pozivu.

```
public class MainActivity extends AppCompatActivity
implements View.OnClickListener{

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button button = findViewById(R.id.button);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        CharSequence text = "Zdravo, ovo je Toast!";
        Toast.makeText(getApplicationContext(), text,
Toast.LENGTH_LONG).show();
    }
}
```

Primer izvršavanja ove aplikacije prikazan je na slici 4.13.

Slika 4.13, primer upotrebe Toast poruke

4.2.8. DatePicker i TimePicker

DatePicker klasa predstavlja vidžet za odabir datuma, i direktno nasleđuje klasu Picker. Analogno, TimePicker klasa predstavlja vidžet za odabir vremena, i takođe nasleđuje klasu Picker. Picker komponente omogućavaju korisniku aplikacije da odabere vrednost iz skupa vrednosti upotrebom pokreta. U slučaju DatePicker i TimePicker komponenti, moguće je pristupiti pojedinačnim delovima datuma i vremena, kao što je prikazano na slici 4.14.

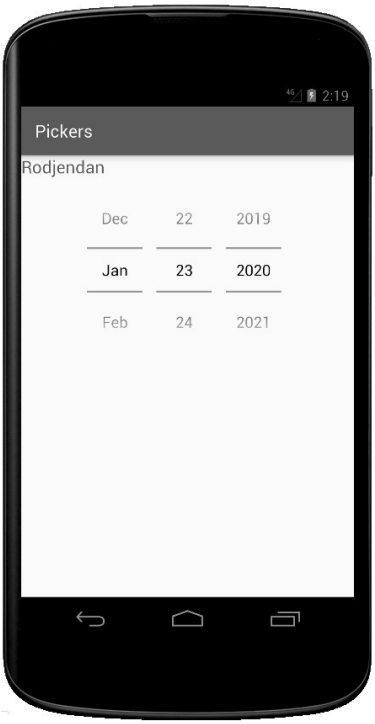
Slika 4.14, TimePicker i DatePicker komponente
(slika preuzeta sa:
https://developer.android.com/guide/topics/ui/controls/pickers)

Svaki Picker pruža kontrole za odabir svakog pojedinačnog dela vremena (sat, minut, AM/PM) ili datuma (mesec, dan, godina). Upotrebom Picker komponenti se osigurava da će korisnik aplikacije odabrati validan datum koji je ispravno formatiran i lokalizovan. Picker se može dodati kroz XML fajl na sledeći način (dat je primer za DatePicker, pošto se sa TimePicker komponentom radi analogno).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView
        android:id="@+id/labelRodjendan"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:text="Rodjendan"/>
    <DatePicker
        android:id="@+id/rodjendan"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:calendarViewShown="false"/>
</LinearLayout>
```

Izgled DatePicker komponente u okviru interfejsa aplikacije dat je na slici 4.15.



Slika 4.15, Izgled DatePicker komponente u okviru aplikacije

Iz aktivnosti koja odgovara datom XML fajlu, DatePicker komponenti se može pristupiti na sledeći način kako bi se, na primer, pojedinačne komponente datuma izvukle i formatirale u String koji se može dalje koristiti u okviru aplikacije:

```
DatePicker rodjendan = (DatePicker)
findViewById(R.id.rodjendan);
String rodjendan = String.format("%4d-%2d-%2d",
rodjendan.getYear(), rodjendan.getMonth(),
rodjendan.getDayOfMonth());
```

Preporučeni način upotrebe DatePicker ili TimePicker komponenti od strane Android zajednice jeste da se koristi DialogFragment kao nosilac ovih komponenti, i nešto je kompleksniji od prethodnog primera. Kako bi se definisao DialogFragment za DatePicker komponentu, neophodno je definisati onCreateDialog() metodu na takav način da vrati instancu klase DatePickerDialog. Nakon toga, potrebno je implementirati interfejs

DatePickerDialog.OnDateSetListener kako bi se dobio callback kada korisnik postavi novi datum. Primer ovakve upotrebe dat je u nastavku kroz Java kod odgovarajućeg fragmenta.

```
public static class DatePickerFragment extends
DialogFragment implements
DatePickerDialog.OnDateSetListener {

    @Override
    public Dialog onCreateDialog(Bundle
    savedInstanceState)
{
        // koristi se trenutni datum kao podrazumevani
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);

        // kreiramo instancu DatePickerDialog i vracamo je
        return new DatePickerDialog(getActivity(), this,
year, month, day);
    }

    public void onDateSet(DatePicker view, int year, int
month, int day) {
        // Uraditi nesto sa datumom koji je korisnik
postavio
    }
}
```

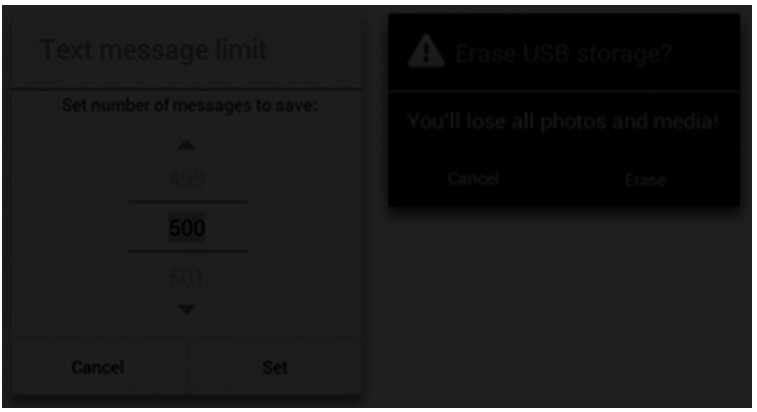
Datum definisan na ovakav način se može prikazati na ekranu pozivom metode:

```
public void showDatePickerDialog(View v) {
    DialogFragment newFragment = new
    DatePickerFragment();
    newFragment.show(getSupportFragmentManager(),
"datePicker");
}
```

4.2.9. Dialog

Dialog komponenta se odnosi na mali prozor koji se prikazuje korisniku i od njega traži da napravi neku odluku ili da unese dodatne informacije. Dialog komponenta ispunjava samo deo ekrana i obično se koristi za modalne događaje koji zahtevaju od korisnika da izvrši neku akciju pre nastavka rada. Primer

Dialog komponenti dat je na slici 4.16.



Slika 4.16, tipični primeri Dialog komponenti
(slika preuzeta sa: https://developer.android.com/guide/topics/ui/dialogs)

Dialog klasa je bazna za sve dijaloge, ali preporuka je da se izbegava direktno instanciranje ove klase. Umesto toga, koriste se izvedene klase poput AlertDialog, ili već pomenutih DatePickerDialog i TimePickerDialog. Sve ove izvedene klase definišu svoj stil i strukturu. Na primer, AlertDialog je komponenta koja se sastoji od naslova, maksimalno tri dugmeta, liste stavki koje se mogu odabrati ili se može koristiti sopstveni layout fajl. U ovom udžbeniku će biti prikazan primer jednostavnog AlertDialoga, dok se za dodavanje sopstvenih dugmića, listi i slično čitalac upućuje na dodatnu literaturu (https://developer.android.com/guide/topics/ui/dialogs).

Još jedna preporuka je da se za svaki Dialog koristi DialogFragment kao nosilac komponente. DialogFragment pruža sve potrebne kontrole za kreiranje Dialog komponente, kao i za njen izgled. Upotrebom DialogFragment klase se osigurava ispravno ponašanje Dialog komponente tokom njenog životnog veka i ispravno rukovanje događajima koje generiše korisnik, poput klika na Back dugme ili rotiranja ekrana. DialogFragment omogućava i da se korisnički interfejs komponente iskoristi kao ubačena komponenta u većem korisničkom interfejsu, kao u slučaju upotrebe tradicionalnih fragmenata.

Primer osnovnog AlertDialog-a, koji koristi DialogFragment kao nosioca, dat je u nastavku. U okviru onCreateDialog metode se koristi AlertDialog.Builder za kreiranje AlertDialog-a, pomoću kojeg je moguće postaviti poruku (setMessage()), kao i dugme sa pozitivnim i negativnim odgovorom (setPositiveButton() i setNegativeButton(), respektivno). Odgovarajuće onClick() metode obezbeđuju callback pozive kada korisnik klikne na neku od ponuđenih opcija.

```
public class FireMissilesDialogFragment extends
DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle
    savedInstanceState)
{
        // Builder klasa se koristi za
udobniju konstrukciju dialoga
        AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
        builder.setMessage("Fire missiles?")
                .setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface
dialog, int id) {
                        // FIRE ZE MISSILES!
                    }
                })
                .setNegativeButton("No", new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface
dialog, int id) {
                        // Korisnik je otkazao dialog
                    }
                });
        // Kreira se AlertDialog objekat I vraca kao
rezultat
        return builder.create();
    }
}
```

Kada želimo da prikažemo naš Dialog, potrebno je napraviti instancu DialogFragment klase i pozvati metodu show(), kojoj se prosleđuju FragmentManager i ime dijaloga kao parametri. FragmentManager se može dohvatiti na dva načina:

- pozivom getSupportFragmentManager() iz aktivnosti, ili
- pozivom getFragmentManager() iz fragmenta.

Primer poziva je dat sa:

```
FireMissilesDialogFragment fragment = new
FireMissilesDialogFragment();
fragment.show(getFragmentManager(), "missiles");
```

Kompletan kod koji ilustruje upotrebu ovako napravljenog AlertDialog-a dat je u nastavku. XML fajl je definisan na sledeći način:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView
        android:id="@+id/labelDialog"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:text="Fire the missiles"/>

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Fire" />
</LinearLayout>
```

Aplikacija sadrži jedno dugme tipa Button, čijim se pritiskom otvara definisani Dialog. U onClick() callback metodi aktivnosti koja odgovara ovom XML fajlu se zbog toga poziva FireMissileDialogFragment. Kod aktivnosti je dat u nastavku, a izgled aplikacije i prikazanog dijaloga nakon pritiska na dugme prikazan je na slici 4.17.

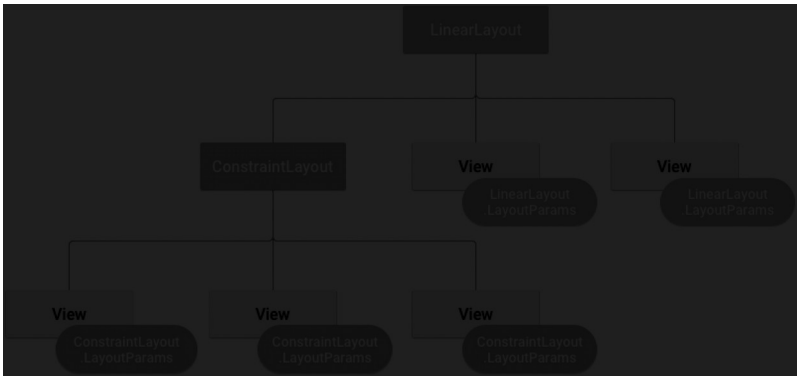
Slika 4.17, prikaz jednostavnog Dialog-a

```
public class MainActivity extends Activity implements
View.OnClickListener{ @Overri
    de
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = findViewById(R.id.button);
        button.setOnClickListener(this);
    }
    public void onClick(View v){
        FireMissilesDialogFragment fragment = new
FireMissilesDialogFragment();
        fragment.show(getFragmentManager(), "missiles");
    }

}
```

4.3. Rasporedi komponenti (layout)

Često je potrebno grupisati komponente grafičkog interfejsa u grupe, kako bi se, na primer, iscrtali na odgovarajući način na ekranu telefona. ViewGroup je specijalna komponenta koja služi kao kontejner za grupisanje drugih komponenti
– unutar ViewGroup komponente se mogu smestiti i View i ViewGroup komponente, koje su u tom slučaju njena deca.

Svaka klasa izvedena iz ViewGroup klase implementira ugnježdenu klasu koja je izvedena iz ViewGroup.LayoutParams. Ova podklasa sadrži atribute koji definišu veličinu i poziciju svakog deteta. Na slici 4.18 se može videti da svaka roditeljska ViewGroup komponenta definiše layout parametre za svako svoje dete (uključujući i druge ViewGroup komponente).



(slika preuzeta sa: https://developer.android.com/guide/topics/ui/declaring-layout)

Slika 4.18, hijerarhija komponenti sa pridruženim layout parametrima

(slika preuzeta sa: https://developer.android.com/guide/topics/ui/declaring-layout)

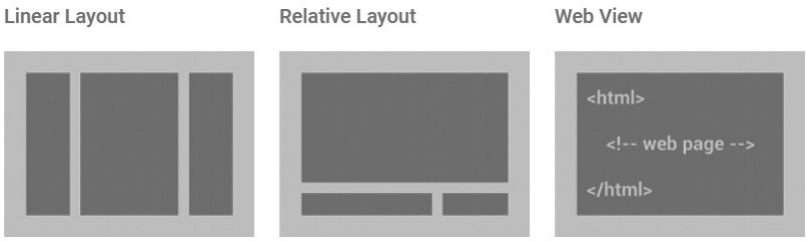
Svaka podklasa klase LayoutParams ima svoju sintaksu za postavljanje vrednosti. Svako dete element mora definisati LayoutParams koji su odgovarajući za njegovog roditelja, a takođe može definisati drugačije LayoutParams za svoju decu koje će oni morati da ispoštuju. Svaki ViewGroup sadrži dva obavezna atributa, širinu i visinu (layout_width i layout_height), a opciono mogu definisati i margine i okvire. Visina i širina se mogu specificirati sa tačnim merama (npr. u pikselima), ali se to ne radi često, već se obično specificiraju jednom od sledeće dve konstante:

- wrap_content – veličina komponente je određena dimenzijama sadržaja te komponente.
- match_parent – komponenta može da postane onoliko velika koliko će joj njena roditeljska komponenta dopustiti.

Uopšteno govoreći, specificiranje širine i visine rasporeda upotrebom apsolutnih jedinica poput piksela nije preporučljivo od strane Android zajednice. Umesto toga, bolji pristup je upotreba relativnih mera poput dp (density-independent pixels), wrap_content ili match_parent. Na taj način se obezbeđuje da se aplikacija iscrta na odgovarajući način na širokom opsegu uređaja sa različitim veličinama ekrana.

Svaka od podklasa klase ViewGroup prikazuje svoju decu na jedinstven način. Najčešće upotrebljavani tipovi rasporeda koji su ugrađeni u Android platformu su (slika 4.19):

- Linearni raspored (LinearLayout) – ređa svoju decu linearno u jedan horizontalni ili vertikalni niz, a ukoliko prozor probije veličinu ekrana automatski kreira scrollbar.
- Relativni raspored (RelativeLayout) – dozvoljava specificiranje lokacije svoje dece relativno jedno u odnosu na drugo (dete A je levo od deteta B) ili u odnosu na roditeljsku komponentu (dete poravnato uz gornju granicu roditeljske komponente).
- Veb raspored (WebView) – služi za prikaz veb stranica.

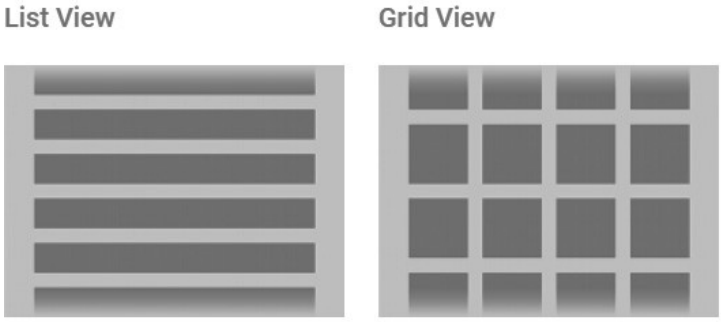


Slika 4.19, najčešće korišćeni rasporedi komponenti

Kada raspored komponenti nije unapred poznat, već se dinamički određuje u vreme izvršavanja, može se koristiti jedan od rasporeda koji je izveden iz bazne klase AdapterView. Ovakvi rasporedi mogu da popune svoj layout sa komponentama u vreme izvršavanja upotrebom Adapter objekta koji služi da poveže podatke sa layout-om. Adapter služi kao komponenta u sredini između izvora podataka i AdapterView rasporeda tako što preuzima podatke (izvor može biti niz podataka, upit u bazu podataka ili podaci dovučeni sa Interneta) i pretvara svaki pojedinačni podatak u View komponentu koja se može dodati u AdapterView. Najčešće upotrebljavani rasporedi koji se zasnivaju na adapteru su (slika 4.20):

- ListView – omogućava dinamičko popunjavanje komponenti koje se smeštaju u listu sa jednom kolonom koja može da se skroluje.
- GridView – omogućava dinamičko popunjavanje komponenti koje se smeštaju u mrežu (sa kolonama i redovima) koja može da se skroluje.

U nastavku ovog poglavlja biće detaljno opisani najčešće korišćeni rasporedi, uz primere koji će ilustrovati tipične upotrebe.



Slika 4.20, najčešće korišćeni rasporedi komponenti koji koriste adapter (slika preuzeta sa: https://developer.android.com/guide/topics/ui/declaring-layout)

4.3.1. Linearni raspored

Linearni raspored (LinearLayout klasa) je raspored koji ređa svoju decu u jednom smeru, vertikalno ili horizontalno (slika 4.21). Smer se definiše atributom android:orientation, koji može da uzme jednu od dve ponuđene vrednosti: horizontal i vertical (podrazumevana vrednost je horizontal).

Slika 4.21, linearni raspored komponenti
(slika preuzeta sa:
https://developer.android.com/guide/topics/ui/layout/linear.html)

Sva deca linearnog rasporeda se postavljaju jedno za drugim, tako da vertikalna lista ima samo jednu kolonu, odnosno po jedno dete u svakom redu (nebitno koliko su pojedina deca široka), a horizontalna lista ima samo jedan red (čija će visina biti određena visinom najvišeg deteta, uz padding). Linearni raspored uračunava i margine između dece, kao i gravity (right, center ili left poravnanje) svakog pojedinačnog deteta.

Linearni raspored dozvoljava dodeljivanje težine svakom pojedinačnom detetu kroz atribut android:layout_weigth. Ovaj atribut zapravo označava koliko je komponenta „bitna“ u smislu koliko prostora će zauzeti na ekranu. Veća vrednost težine dozvoljava komponenti da se proširi kako bi ispunila nepopunjen prostor u roditeljskoj komponenti. Svako dete može specificirati težinu, a onda će preostali dostupni prostor biti proporcionalno podeljen komponentama na osnovu njihove deklarisane težine. Podrazumevana vrednost ovog atributa je 0.

Na primer, ukoliko se želi linearan raspored u kome svako dete komponenta uzima istu količinu prostora na ekranu, potrebno je postaviti atribut android:layout_height svakog deteta na 0dp (za vertikalni raspored), odnosno android:layout_width svakog deteta na 0dp (za horizontalni raspored), a nakon toga postaviti težinu svakog deteta postavljanjem atributa android:layout_weight na 1.

Ukoliko je potrebna nejednaka raspodela, moguće je napraviti linearni raspored na takav način da različita deca uzimaju različite veličine prostora na ekranu. Na primer:

- U rasporedu postoje tri TextView komponente, pri čemu prve dve deklarišu težinu 1, a trećoj nije dodeljena nikakva težina. U tom slučaju, treći TextView koji je bez dodeljene težine neće moći da raste. Zapravo,

on će zauzimati samo onoliko prostora koliko zahteva njegov sadržaj. Druge dve TextView komponente će se podjednako proširiti kako bi ispunile kompletan prostor koji preostaje nakon merenja sve tri komponente.

- Ukoliko prve dve komponente tipa TextView deklarišu težinu 1, a treća TextView komponenta deklariše težinu 2 (umesto 0), ona se posmatra kao važnija komponenta u poređenju sa prve dve, i preuzima polovinu preostalog prostora. Prve dve komponente dele ostatak prostora ravnomerno.

Kako bismo ilustrovali rad sa težinama, neka se posmatra konkretan primer gde je potrebno razviti korisnički ekran u obliku vertikalnog linearnog rasporeda koji služi u aktivnosti za slanje poruke u okviru neke aplikacije. Neka je potrebno da postoje polja **Primalac**, **Naslov** i dugme **Pošalji** koji uzimaju onoliko prostora koliko im je potrebno. Preostali prostor visine ekrana ove aktivnosti treba da bude rezervisan za samu poruku. XML fajl koji ispunjava ove zahteve dat je u nastavku. Posebna pažnja treba da se obrati na EditText koji se koristi za unos teksta poruke. Pošto se želi da on zauzme kompletan preostali prostor koji ostane nakon merenja potrebnog prostora svih komponenti, potrebno je navesti da atribut android:layout_height bude postavljen na vrednost 0dp, a android:layout_weight bude postavljen na vrednost 1.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="10dp"
    android:paddingRight="10dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/primalac" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/naslov" />
    <EditText
        android:layout_width="match_parent"
        android:layout_h eight="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/tekst" />
    <Button
```

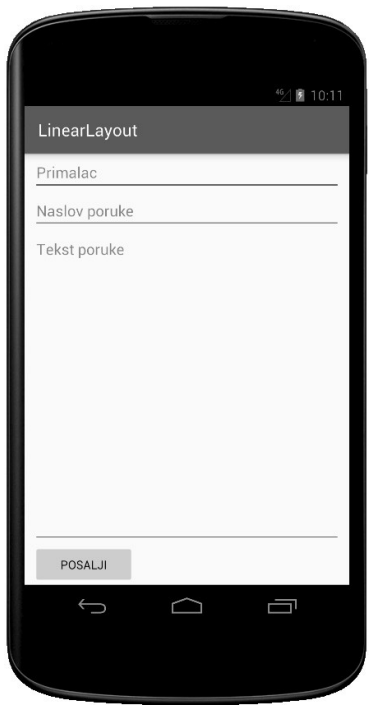
```
        android:layout_width="120dp"
```

```
        android:layout_height="wrap_content"
        android:layout_gravity="left"
        android:text="@string/posalji" />
</LinearLayout>
```

Ovaj primer podrazumeva da su Stringovi koji se koriste nisu hard-kodovani već da su definisani u strings.xml fajlu (preporučeni način, koji olakšava održavanje i lokalizaciju aplikacije). Sadržaj strings.xml fajla:

```
<resources>
    <string name="app_name">LinearLayout</string>
    <string name="primalac">Primalac</string>
    <string name="naslov">Naslov poruke</string>
    <string name="tekst">Tekst poruke</string>
    <string name="posalji">Posalji</string>
</resources>
```

Izgled rezultujućeg ekrana dat je na slici 4.22.



Slika 4.22, primer vertikalnog linearnog rasporeda sa težinama

4.3.2. Relativni raspored

Relativni raspored (RelativeLayout) je raspored koji svoju decu postavlja na relativne pozicije (slika 4.23). Pozicija svakog deteta može se postaviti relativno u odnosu na neku drugu komponentu dete (poput levo ili ispod druge komponente) ili relativno u odnosu na roditeljsku RelativeLayout komponentu (na primer poravnato sa dnom, levo, desno ili centrirano).

Slika 4.23, relativni raspored komponenti
(slika preuzeta sa:
https://developer.android.com/guide/topics/ui/layout/relative.html)

Relativni raspored je veoma moćan alat za dizajn korisničkog interfejsa pošto eliminiše potrebu za ugnježdenim linearnim rasporedima, čime se hijerarhija rasporeda održava ravnom (što može drastično uticati na performanse). Opšte pravilo je da ukoliko je za pravljenje zahtevanog rasporeda potrebno više ugnježdenih linearnih rasporeda, treba razmisliti i zameniti ih sa jednim relativnim rasporedom.

U relativnom rasporedu komponenta se pozicionira relativno u odnosu na roditeljsku komponentu ili u odnosu na neku drugu komponentu, pri čemu se komponente referenciraju putem svog ID. Dakle, moguće je poravnati nekoliko komponenti uz levu ivicu, ili staviti jednu komponentu ispod druge, centrirati komponentu i slično. Podrazumevano se sve komponente iscrtavaju u gornjem levom uglu rasporeda, pa je neophodno definisati poziciju svake komponente kako se ne bi iscrtavale jedna preko druge. Za ovo se koriste atributi definisani u RelativeLayout.LayoutParams. U ovoj klasi postoji veliki broj parametara, od kojih su neki:

- android:layout_alignParentTop – true vrednost ovog atributa će poravnati gornju ivicu ove komponente sa gornjom ivicom roditeljske komponente.

- android:layout_centerVertical – true vrednost ovog atributa će centrirati ovu komponentu vertikalno u okviru roditeljske komponente.
- android:layout_below – poravnava gornju granicu ove komponente ispod komponente referencirane sa svojim ID.
- android:layout_toRightOf – postavlja levu ivicu ove komponente desno od komponente referencirane sa svojim ID.

Detaljna lista svih opcija za relativno pozicioniranje je data u tabeli 4-3 (izvor https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html).

XML atributi	Opis
android:layout_above	Positions the bottom edge of this view above the given anchor view ID.
android:layout_alignBaseline	Positions the baseline of this view on the baseline of the given anchor view ID.
android:layout_alignBottom	Makes the bottom edge of this view match the bottom edge of the given anchor view ID.
android:layout_alignEnd	Makes the end edge of this view match the end edge of the given anchor view ID.
android:layout_alignLeft	Makes the left edge of this view match the left edge of the given anchor view ID.
android:layout_alignParentBottom	If true, makes the bottom edge of this view match the bottom edge of the parent.
android:layout_alignParentEnd	If true, makes the end edge of this view match the end edge of the parent.
android:layout_alignParentLeft	If true, makes the left edge of this view match the left edge of the parent.
android:layout_alignParentRight	If true, makes the right edge of this view match the right edge of the parent.
android:layout_alignParentStart	If true, makes the start edge of this view match the start edge of the parent.
android:layout_alignParentTop	If true, makes the top edge of this view match the top edge of the parent.
android:layout_alignRight	Makes the right edge of this view match the right edge of the given anchor view ID.
android:layout_alignStart	Makes the start edge of this view match the start edge of the given anchor view ID.
android:layout_alignTop	Makes the top edge of this view match the top edge of the given anchor view ID.

android:layout_alignWithParentIfMissing	If set to true, the parent will be used as the anchor when the anchor cannot be be found for layout_toLeftOf, layout_toRightOf, etc.
android:layout_below	Positions the top edge of this view below the given anchor view ID.
android:layout_centerHorizontal	If true, centers this child horizontally within its parent.
android:layout_centerInParent	If true, centers this child horizontally and vertically within its parent.
android:layout_centerVertical	If true, centers this child vertically within its parent.
android:layout_toEndOf	Positions the start edge of this view to the end of the given anchor view ID.
android:layout_toLeftOf	Positions the right edge of this view to the left of the given anchor view ID.
android:layout_toRightOf	Positions the left edge of this view to the right of the given anchor view ID.
android:layout_toStartOf	Positions the end edge of this view to the start of the given anchor view ID.

Tabela 4-3, lista svih atributa za relativno pozicioniranje komponenti

Jedna stvar na koju je potrebno obratiti pažnju jeste da se ne dođe do situacije da se napravi tzv. kružna zavisnost tokom definisanja relativnih pozicija komponenti, odnosno da se komponenta A postavi u odnosu na komponentu B, a komponenta B se postavi u odnosu na komponentu A, što nije dozvoljeno. Dodatno, može da se desi da ukoliko se napravi greška u definisanju relativnih pozicija komponenti da jedna komponenta zakloni drugu komponentu. Na primer, ako su dve View komponente istovremeno poravnate sa gornjom i levom ivicom roditeljske komponente, druga komponenta će zakloniti prvu.

Tipičan primer upotrebe relativnog rasporeda prikazan je kroz implementaciju ekrana koji predstavlja login formu. Moguće je uočiti da XML teži da bude značajno veći nego u slučaju linearnog rasporeda, zbog potrebe za definisanjem relacija između komponenti. XML je dat u sledećem isečku koda:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/labelLogin"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerInParent="false"
        android:text="@string/login"
        android:textSize="20dp"
        android:layout_marginTop="5dp" />
    <TextView
        android:id="@+id/labelUsername"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/labelLogin"
        android:layout_marginTop="20dp"
        android:text="@string/username"
        android:textSize="22dp" />

    <EditText
        android:id="@+id/inputUsername"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignTop="@+id/labelUsername"
        android:layout_toRightOf="@+id/labelUsername" />
    <TextView
        android:id="@+id/labelPassword"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/labelUsername"
        android:layout_marginTop="20dp"
        android:text="@string/password"
        android:textSize="22dp" />
    <EditText
        android:id="@+id/inputPassword"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignTop="@+id/labelPassword"
        android:layout_toRightOf="@+id/labelPassword"
        android:inputType="textPassword" />
    <Button
        android:id="@+id/btnSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="false"
        android:layout_below="@+id/inputPassword"
        android:layout_centerInParent="true"
        android:text="@string/submit" />
    <Button
        android:id="@+id/btnSignUp"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="@string/signup"
        android:layout_centerHorizontal="true"/>
</RelativeLayout>
```

Stringovi su definisani u res/values folderu, u fajlu strings.xml, čiji je sadržaj dat u nastavku. Izgled rezultujućeg ekrana aplikacije prikazan je na slici 4.24.

```
<resources>
    <string name="app_name">RelativeLayout</string>
    <string name="login">LOGIN</string>
    <string name="username">Username :</string>
    <string name="password">Password :</string>
    <string name="submit">Submit</string>
    <string name="signup">Sign up</string>
</resources>
```

Slika 4.24, primer relativnog rasporeda

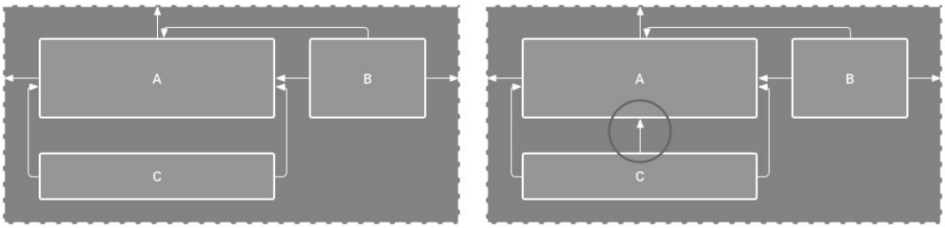
4.3.3. Ograničeni raspored

Ograničeni raspored (ConstraintLayout) je raspored komponenti veoma sličan relativnom rasporedu, sa tom razlikom što je fleksibilniji. Prema najavama najnovije verzije Androida, ovaj raspored spada u deo Androidovih biblioteka koje se više ne razvijaju, odnosno, neće više biti podržan dolaskom AndroidX platforme. Inicijalno, ConstraintLayout je dozvoljavao kreiranje velikih i kompleksnih rasporeda sa ravnom hijerarhijom (bez ugnježđavanja, kao kod relativnog rasporeda). Komponente se postavljaju na osnovu odnosa između samih komponenti i odnosa sa roditeljskom komponentom.

Osim što je nešto fleksibilniji od relativnog rasporeda, glavna prednost ograničenog rasporeda je direktna dostupnost kroz Layout Editor alat (grafički editor za kreiranje rasporeda komponenti), pošto su specijalno razvijeni jedno za drugo. Kada se kreira raspored komponenti pomoću ConstraintLayout, u potpunosti je moguće razviti ga pomoću drag & drop-a komponenti iz paleta, bez pisanja XML koda.

Kako bi se definisala pozicija komponente u ograničenom rasporedu, potrebno je dodati najmanje jedno horizontalno i jedno vertikalno ograničenje za tu komponentu. Svako ograničenje predstavlja konekciju ili poravnanje sa drugom komponentom, roditeljskim rasporedom ili sa nevidljivom linijom vodiljom. Svako ograničenje definiše poziciju komponente na vertikalnoj i horizontalnoj osi, tako da je minimum potrebnih ograničenja jedno po osi, a vrlo često je potrebno i više. Kada se komponenta prevuče iz palete na ekran u grafičkom editoru, ona ostaje pozicionirana tamo gde je spuštena čak i ako nema ograničenja. Međutim, to je učinjeno samo da bi se olakšalo dalje modifikovanje. Ako komponenta nema ograničenja, prilikom izvršavanja na fizičkom uređaju biće iscrtana na poziciji (0,0), odnosno u gornjem levom uglu ekrana.

Raspored komponenti sa leve strane slike 4.25 izgleda dobro u grafičkom editoru, međutim, ne postoji vertikalno ograničenje za komponentu C (ograničena je samo horizontalno tako da bude poravnata sa komponentom A). Prilikom crtanja ovog rasporeda na fizičkom uređaju, komponenta C će biti poravnata po levoj i desnoj ivici sa komponentom A, ali će biti prikazana na vrhu ekrana jer nema vertikalno ograničenje. Raspored komponenti sa desne strane slike 4.25 će biti ispravno prikazan na fizičkom uređaju, pošto postoji i vertikalno ograničenje komponente C u odnosu na komponentu A. Ukoliko neko ograničenje nedostaje, to neće izazvati grešku pri kompajliranju, međutim, biće označeno u grafičkom editoru kao greška. Grafički editor se može podesiti tako da automatski doda ograničenja prilikom pozicioniranja komponenti na ekranu.



Slika 4.25, dodavanje ograničenja komponenti u ograničenom rasporedu (izvor: https://developer.android.com/training/constraint-layout)

4.3.4. ListView

ListView raspored prikazuje vertikalnu kolekciju komponenti kroz koju je moguće skrolovati, pri čemu se svaka komponenta pozicionira neposredno ispod prethodne komponente u listi. Postoji i fleksibilnija opcija, RecyclerView (uz ograničenje da RecyclerView neće biti podržan od strane AndroidX platforme).

ListView raspored ćemo opisati na tipičnom primeru. Potrebno je prikazati listu predmeta na fakultetu, koji su za potrebe primera smešteni u niz (mogu se dovlačiti i sa Interneta ili iz baze). Pošto se potrebni podaci nalaze u nizu, može se koristiti ArrayAdapter. ArrayAdapter podrazumevano kreira komponentu za svaki element niza tako što poziva njegovu toString() metodu, i smešta dobijeni sadržaj u TextView. ArrayAdapter se inicijalizuje na sledeći način:

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,
        R.layout.list_item, nizKurseva);
```

Parametri koji se prosleđuju konstruktoru su kontekst aplikacije (u većini slučajeva može se proslediti this), XML fajl sa rasporedom komponenti koji obezbeđuje prikaz pojedinačnog elementa iz niza, a poslednji argument je niz Stringova odakle će se izvlačiti podaci za popunjavanje svakog pojedinačnog TextView elementa.

ListView raspored se jednostavno deklariše u fajlu sa XML rasporedom. Sadržaj fajla activity_main.xml:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <ListView
        android:id="@+id/listaPredmeta "
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content" >
    </ListView>
</LinearLayout>
```

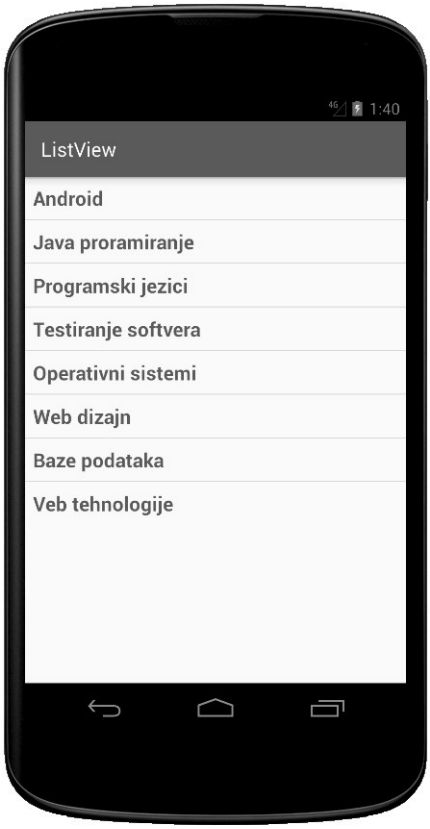
Potrebno je dodati još jedan fajl u res/layout direktorijum, gde će biti opisan raspored jedne komponente liste za prikaz. Dizajn pojedinačne komponente ListView rasporeda dat je fajlom list_item.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/label"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="8dp"
    android:textSize="20dp"
    android:textStyle="bold" >
</TextView>
```

Kod aktivnosti MainActivity.java:

```
package com.example.student.listview;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {

    // Niz kurseva na fakultetu...
    String[] nizKurseva = {"Android","Java
proramiranje","Programski jezici","Testiranje softvera",
            "Operativni sistemi","Web dizajn","Baze
podataka","Veb tehnologije"};

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ArrayAdapter adapter = new
ArrayAdapter<String>(this,
                R.layout.list_item, nizKurseva);

        ListView listView =
findViewById(R.id.listaPredmeta);
        listView.setAdapter(adapter);
    }
}
```

Nakon pokretanja aplikacije, njen prikaz na ekranu dat je na slici 4.26.



Slika 4.26, tipičan primer upotrebe ListView rasporeda komponenti

4.4. Layout editor

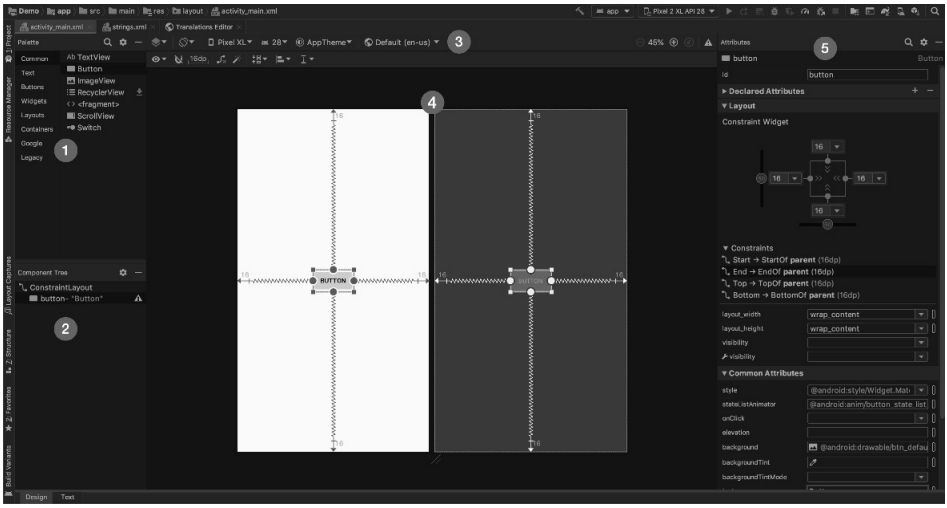
Android Studio sadrži grafički editor za dizajniranje korisničkog interfejsa, pod nazivom Layout Editor. Upotrebom editora može se veoma brzo napraviti raspored komponenti na ekranu, prostim prevlačenjem komponenti iz palete u radni prostor editora, čime se skoro u potpunosti eliminiše potreba da se komponente definišu ručno pisanjem XML fajla. Editor omogućava i trenutni pregled rasporeda, kao i prikaz na više različitih verzija Android uređaja i veličina ekrana, kako bi se raspored mogao dinamički prilagoditi. Na taj način se osigurava da će dizajn raditi dobro na različitim ekranima.

LayoutEditor se može koristiti sa svim rasporedima komponenti. Optimizovan je inicijalno za ograničeni raspored, međutim sasvim dobro radi i sa relativnim i

sa

linearnim rasporedima komponenti. Editor će se automatski pojaviti čim se otvori XML fajl sa rasporedom komponenti. Njegove glavne komponente su prikazane na slici 4.27 (izvor: https://developer.android.com/studio/write/layout- editor.html#intro):

1. Paleta – lista komponenti (View i ViewGroup) koje se mogu prevući na raspored.
2. Stablo komponenti – prikazuje hijerarhijski pogled na trenutni raspored.
3. Alati – skup opcija za konfiguraciju prikaza rasporeda u editoru.
4. Dizajn editor – raspored prikazan kao dizajn, plan ili oba istovremeno.
5. Atributi – prikaz svih atributa trenutno odabrane komponente, koji se mogu modifikovati po potrebi.



Slika 4.27, Layout Editor - grafički editor za dizajn rasporeda komponenti

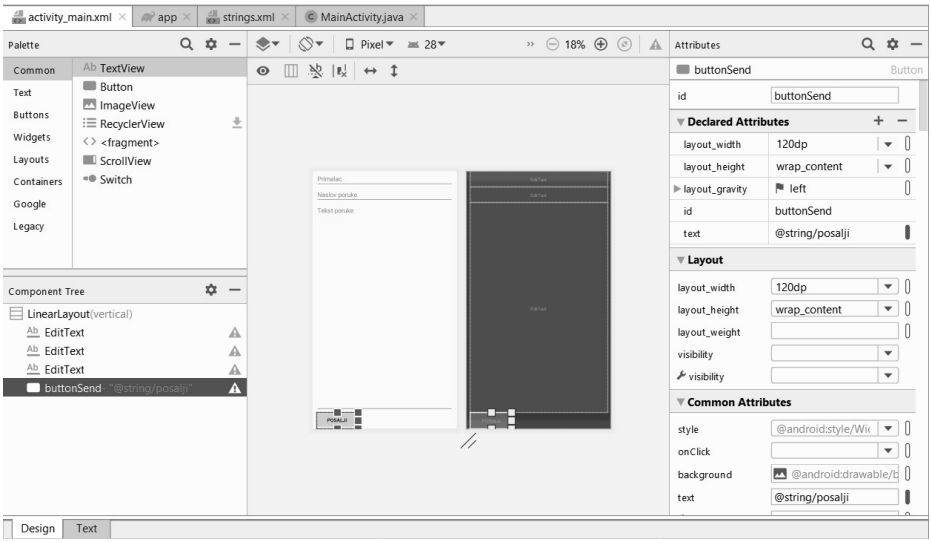
Moguće je i dalje editovati XML kod. Ukoliko je potrebno prikazati XML kod, potrebno je odabrati Text tab na dnu prozora. Prozor sa atributima nije vidljiv dok se programer nalazi u tekstualnom editoru XML fajla.

Prilikom dodavanja novog rasporeda za aplikaciju, proces počinje dodavanjem novog XML fajla u res/layout direktorijum. Nakon toga se komponente dodaju prostim prevlačenjem iz palete komponenti. Atributi za svaku komponentu se mogu videti u prozoru Attributes, koji se nalazi sa desne strane editora (slika 4.28).

Atributi su podeljeni u nekoliko sekcija. Sekcija Declared Attributes se odnosi na atribute koji su već specificirani u rasporedu. Sekcija Layout se odnosi na atribute koji se odnose na prikaz komponente, odnosno njenu širinu, visinu,

vidljivost i

slično. Nakon toga se nalazi sekcija Common Attributes koja obuhvata izdvojeno najčešće korišćene atribute za taj tip komponente. Na kraju, nalazi se sekcija All Attributes u kojoj su izlistani svi atributi komponente (ima ih jako veliki broj, zbog toga je i izdvojena sekcija Common Attributes gde je programeru koji još uvek nije naviknut na okruženje olakšano pronalaženje najbitnijih atributa). Svaka promena atributa se automatski unosi u XML kod rasporeda na kojem se trenutno radi.



Slika 4.28, podešavanje atributa komponente u grafičkom editoru

Uvođenje grafičkog editora u Android Studio znatno je olakšalo razvoj korisničkog interfejsa i drastično smanjilo potrebu za pisanjem XML fajlova. Međutim, i dalje je potrebno poznavati strukturu XML fajla, posebno zbog činjenice da nisu sve komponente dostupne u paleti. Na primer, DatePicker komponenta se ne nalazi u paleti, pa se njeno dodavanje u raspored mora odraditi ručno kroz XML kod.

4.5. Klasa R

Neki atributi su zajednički za sve komponente (View objekte). To su atributi definisani u baznoj klasi View, i koje nasleđuju sve izvedene klase, od kojih je najbitniji atribut id. To praktično znači da sve komponente grafičkog interfejsa imaju svoj id.

Kada se aplikacija kompajlira, svakom View objektu se dodeljuje jedinstvena celobrojna vrednost koja se koristi za identifikaciju, i smešta u poseban fajl po

imenu R.java. Ovaj fajl se automatski generiše prilikom build-a aplikacije, i smešta ove jedinstvene identifikatore u posebne grupe: drawables, strings, layout itd. Osnovna svrha ove klase je omogućavanje brzog pristupa resursima u projektu. Prilikom dodavanja novih resursa ili brisanja postojećih resursa, R.java fajl će se automatski ažurirati.

Svaka komponenta korisničkog interfejsa je definisana u XML fajlu. Na primer, posmatrajmo dugme definisano sledećim isečkom XML koda:

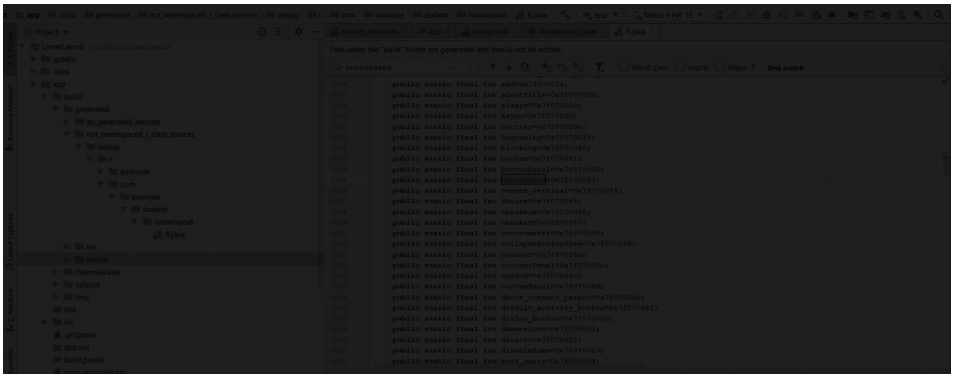
```
<Button
    android:id="@+id/buttonSend"
    android:layout_width="120dp"
    android:layout_height="wrap_content"
    android:layout_gravity="left"
    android:text="@string/posalji" />
```

Simbol @ na početku stringa obaveštava XML parser da treba da parsira ostatak Id stringa i da ga identifikuje kao Id resursa. Simbol + označava da je ovo novi resurs sa novim imenom koje se mora dodati u resurse aplikacije, odnosno u R.java fajl.

Kada je View jedinstveno identifikovan, može se referencirati iz Java izvornog koda. Na primer, dugme definisano u prethodnom XML kodu ima dodeljen Id "buttonSend", može se dohvatiti iz Java koda pozivom metode findViewById(). Ova metoda locira komponentu preko Id definisanog u XML fajlu, i koristi se na sledeći način:

```
Button button = findViewById(R.id.buttonSend);
```

Svi resursi se mogu adresirati na sličan način. Prilikom kompajliranja, oni se smeštaju u R.java klasu, koja se tipično nalazi u direktorijumu app/build/generated. Ovaj fajl nije preporučljivo ručno menjati. Primer sadržaja R.java klase (sa označenim buttonSend Id) dat je na slici 4.29.



Slika 4.29, R.java klasa, lokacija i njen sadržaj

4.6. Rukovanje događajima

Android nudi više načina za prihvatanje događaja koje korisnik generiše u interakciji sa aplikacijom. Svaki View objekat može generisati događaje koje je moguće kasnije obraditi u kodu aplikacije. Prilikom razvijanja interaktivnosti korisničkog interfejsa, uobičajen pristup je da se prihvate događaji od tačno određene View komponente nad kojom korisnik vrši interakciju.

Ukoliko se detaljnije pogleda kod različitih View klasa od kojih se pravi korisnički interfejs, moguće je primetiti nekoliko callback metoda koje su veoma korisne prilikom obrade korisničkih događaja. Ove metode poziva Android sistem kada se odgovarajuća akcija nad objektom izvrši. Na primer, kada se neki View (poput Button ili EditText) dodirne, biće pozvana onTouchEvent() metoda nad tim objektom. Kako bi se omogućilo programeru da reaguje na događaje, klasa View sadrži kolekciju interfejsa sa callback metodama koje se lako mogu implementirati. Ovi interfejsi su poznati pod zajedničkim imenom – osluškivači događaja (engl. *event listeners*),

Osluškivač događaja je dakle interfejs u View klasi koji sadrži jednu callback metodu. Na primer, OnClick osluškivač je unutar View klase definisan na sledeći način (preuzeto iz izvornog koda Android platforme):

```
/**
      * Interface definition for a callback to be invoked
when a view is clicked.
           */
      public interface OnClickListener {
          /**
           * Called when a view has been clicked.
           *
           * @param v The view that was clicked.
           */
           void onClick(View v);
      }
```

Ovu metodu će pozvati Android Sistem u onom trenutku kada View objekat na kog je osluškivač registrovan dobije odgovarajući događaj od korisnika. Konkretni View objekat koji je generisao događaj se prosleđuje kao parametar metode. Najčešće korišćeni osluškivači, odnosno callback metode su:

- onClick() – nalazi se u interfejsu View.OnClickListener, poziva se kada korisnik dodirne kopmonentu.
- onLongClick() – nalazi se u interfejsu View.OnLongClickListener, poziva se kada korisnik dodirne i zadrži dodir na komponenti.

- onFocusChange() – nalazi se u interfejsu View.OnFocusChangeListener, poziva se kada korisnik navigira do ili od komponente upotrebom tastera za navigaciju.
- onKey() – nalazi se u interfejsu View.OnKeyListener, poziva se kada se korisnik fokusira na komponentu i pritisne/otpusti hardversko dugme.
- onTouch() – nalazi se u interfejsu View.OnTouchListener, poziva se kada korisnik izvrši akciju koja se može okvalifikovati kao dodir, na primer, pritisak na komponentu, otpuštanje komponente ili bilo koji pokret na ekranu unutar granica komponente.

Ove metode su jedini članovi odgovarajućih interfejsa. Ako se posmatraju povratne vrednosti ovih metoda, onClick() na primer je tipa void, odnosno nema povratnu vrednost. Sa druge strane, onLongClick() vraća vrednost tipa boolean, koja označava da li događaj treba da se obrađuje i dalje ili ne. Ako je vraćena vrednost true, time se označava da je događaj u potpunosti obrađen i zaustavlja se svaka dalja obrada. U suprotnom, vrednost false označava da događaj nije obrađen, odnosno da drugi osluškivači treba da preuzmu obradu.

Logika implementacije obrade događaja je slična za sve interfejse. Postoje dva osnovna pristupa pri implementaciji metode i obradi događaja. Oba su pokazana u nastavku, na primeru registrovanja osluškivača za onClick na komponenti tipa Button. Prvi pristup podrazumeva upotrebu anonimne klase, kroz čiju se direktnu definiciju daje i implementacija metode onClick(). Ovaj pristup prikazan je kroz sledeći isečak aktivnosti:

```
Button button = findViewById(R.id.buttonSend);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // dugme je pritisnuto, ovde treba uraditi nesto
    }
});
```

Često je bolji i fleksibilniji pristup da se interfejs OnClickListener implementira na nivou aktivnosti, čime se izbegava dodatno učitavanje klasa i alokacija objekata. On podrazumeva da aktivnost mora da implementira metodu onClick() nasleđenu iz interfejsa, a pri registraciji osluškivača aktivnost može proslediti samu sebe. Ovaj pristup dat je u primeru koda sledeće aktivnosti.

```
public class MainActivity extends AppCompatActivity
implements View.OnClickListener{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        Button button = findViewById(R.id.buttonSend);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        // dugmme je kliknuto, ovde treba uraditi nesto
    }
}
```

4.7. Pitanja za vežbu

- Koja je osnovna klasa u hijerarhiji kompomenti grafičkog korisničkog interfejsa?
- Šta je dp?
- Šta je gustina ekrana?
- Koji je odnos nasleđivanja imeđu klasa View i ViewGroup?
- Kojoj klasi pripadaju svi rasporedi komponenti u Androidu?
- Koji sve različiti rasporedi komponenti postoje?
- Kako se elementiređaju unutar Linear Layout komponente sa vertikalnom orijentacijom?
- Kako se zove komponenta koja se koristi za grupisanje elemenata jedan ispod drugog?
- Kako se sve mogu definisati pozicije komponenti u okviru relativnog rasporeda komponenti?
- Šta je to ograničeni raspored? Kako se u njemu postavljaju komponente?
- Po čemu se ograničeni raspored razlikuje od relativnog?
- Šta je ListView?
- U kom resursnom fajlu se definiše koje su komponente vidljive u korisničkom interfejsu i kakav je njihov raspored?
- Koliko korenih elemenata sme imati resursni fajl koji definiše raspored komponenti?
- Kako se zove funkcija koja se koristi da u Java kodu referencira element iz grafičkog interfejsa?
- Čemu služi atribut id?
- Koji sve još atributi postoje za različite komponente grafičkog interfejsa?
- Kojim atributima se kontrolišu dimenzije komponente?
- Koji osluškivač (listener) se dodeljuje dugmetu kako bi se upravljalo Click događajem nad njim?
- Koji još tipovi osluškivača (listenera) postoje?
- U Android aplikaciji potrebno je prikazati više radio dugmadi od kojih korisnik bira samo jedno. Koju komponentu treba koristiti za grupisanje radio dugmadi?
- Čemu služi klasa R.java u Androidu?
- Koje sve standardne komponente grafičkog interfejsa postoje?
- Šta je Toast?
- Šta je Spinner?
- Šta su adapteri i kako se koriste?
- Koje Picker komponente postoje?
- Kako se koriste dijalozi?

- Kako se zove grafički editor u kome se definiše raspored korisničkog interfejsa?
- Kako se može podesiti koju softversku tastaturu će Android izbaciti kada korisnik uđe u polje za unos teksta? Koje sve tastature postoje?
- Kako se u opštem slučaju rukuje događajima korisničkog interfejsa?

5. Aktivnosti

Activity klasa je ključna komponenta Android aplikacije, a način na koji se aktivnosti pokreću je jedan od osnovnih koncepata modela aplikacije na Android platformi. U klasičnom Java programiranju, aplikacija se pokreće pokretanjem main() metode (koju poziva JVM), dok se u Androidu aktivnost pokreće pozivom jedne od specifičnih callback metoda koje odgovaraju različitim fazama životnog ciklusa aktivnosti.

Mobilne aplikacije se razlikuju od tradicionalnih desktop aplikacija i po tome što mobilna aplikacija ne mora uvek da započne na istom mestu, za razliku od desktop aplikacija koje obično imaju jednu tačno definisanu tačku ulaska (main() metoda). Kako bismo ilustrovali ovaj slučaj, posmatrajmo primer mejl aplikacije. Ukoliko se ovoj aplikaciji pristupi sa glavnog ekrana Androida pokreće se standardan korisnički interfejs sa prikazom liste mejlova. Međutim, ukoliko se iz neke druge aplikacije pokrene mejl aplikacija, moguće je da će se direktno otići na ekran za sastavljanje i slanje mejla. Još jedan tipičan primer je da kada direktno otvorimo neku aplikaciju socijalne mreže, videćemo feed sa novostima i statusima naših kontakata. Ukoliko se iz neke druge aplikacije, na primer galerije sa slikama, uradi share neke slike na tu socijalnu mrežu, direktno se ide na share ekran te aplikacije.

Razlog tome je u samom konceptu Android platforme. Kada iz jedne aplikacije zovemo drugu aplikaciju, ne pozivamo zapravo drugu aplikaciju ako atomsku celinu, već pozivamo jednu njenu aktivnost. To znači da aktivnost zapravo služi kao tačka ulaska u aplikaciju.

Najjednostavnije rečeno, aktivnost obezbeđuje jedan prozor po kome aplikacija može da iscrta svoj korisnički interfejs. Tipično, ovaj prozor ispunjava ceo ekran, ali može biti i manji od ekrana. Osnovni princip kreiranja Android aplikacija se svodi na to da jedna aktivnost implementira jedan ekran u aplikaciji. Aplikacija se može sastojati od više ekrana, a u tom slučaju će postojati aktivnost za svaki od pojedinačnih ekrana. Na primer, jedna aktivnost aplikacije može da traži logovanje korisnika na sistem, druga aktivnost može prikazati listu fotografija, treća aktivnost može biti ekran sa podešavanjima aplikacije, a četvrta aktivnost da omogući odabir pojedinačne fotografije.

Jedna aktivnost u aplikaciji se obično označi kao MainActivity (analogno sa main() metodom u tradicionalnim Java programima) – ova aktivnost odgovara prvom ekranu koji se pojavi kada korisnik pokrene aplikaciju. Nakon toga, ova aktivnost može pozvati neku drugu, ta druga aktivnost neku treću, kako bi se izvršile akcije koje korisnik zahteva. Ako se opet vratimo na primer jednostavne mejl aplikacije, MainActivity bi mogao da pruži osnovni ekran koji pokazuje prijemno sanduče sa listom primljenih mejlova. Iz ove aktivnosti, korisnik može

da pokrene druge aktivnosti, kako bi dobio druge ekrane na kojima može da izvrši akcije, poput ekrana za detaljan pregled pojedinačnog mejla ili ekrana za pisanje novog mejla. Aktivnosti jedne aplikacije rade zajedno kako bi formirali tečan i smislen korisnički interfejs, ali pojedinačne aktivnosti su samo labavo povezane jedne sa drugima. Labava sprega znači da postoji vrlo malo zavisnosti između pojedinačnih aktivnosti.

5.1. Konfigurisanje manifesta

Aktivnosti, zajedno sa određenim atributima, moraju biti deklarisane u manifestu aplikacije. Deklaracija se vrši u AndroidManifest.xml fajlu, dodavanjem
<activity> elementa kao deteta <application> elementa. Primer je dat sledećim isečkom XML koda AndroidManifest fajla, koji deklariše aktivnost pod nazivom MainActivity:

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action
android:name="android.intent.action.MAIN" />

            <category
android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
android:name=".ConfirmationActivity"></activity>
</application>
```

Iz prethodnog primera se vidi da se osim activity elementa koristi još jedan, pod nazivom intent filter (filter namera). Intent filteri su veoma moćan alat Android platforme, koji omogućava da se aktivnost startuje ne samo na osnovu eksplicitnog zahteva, već i na osnovu implicitnog. Eksplicitna namera je kada se sistemu direktno kaže da želimo da odemo u Send Email aktivnost Gmail aplikacije. Dakle, precizno je definisano koja tačno aktivnost treba da se pokrene. Nasuprot tome, implicitni zahtev bi bio da kažemo da želimo da pošaljemo mejl i da tražimo od sistema da nam startuje Send Email ekran bilo koje aktivnosti koja može da nam završi posao.

5.2. Životni ciklus aplikacije

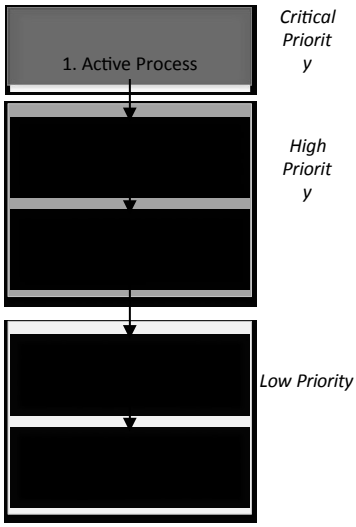
Kako bismo razumeli životni ciklus pojedinačnih aktivnosti, najpre je potrebno shvatiti životni ciklus aplikacije. Ovde prvi put jasno dolazi do izražaja razlika u programiranju klasičnih aplikacija za desktop računare i aplikacija za mobilne uređaje. Mobilni uređaj je veoma ograničen po pitanju resursa – nedovoljno memorije, limitirana procesorska moć, kao i ograničenja po pitanju energije (rad sa baterijom). Implikacija koje programer mora biti svestan u svakom trenutku razvoja aplikacije - kako bi upravljao ograničenim sistemskim resursima na što optimalniji način, Android sistem može da ubije bilo koju aplikaciju koja se izvršava.

Neka se sada posmatra sam način izvršavanja aplikacije pod Android sistemom. U najvećem broju slučajeva, svaka aplikacija se izvršava u svom Linux procesu. Proces se kreira u trenutku kada je potrebno izvršiti neki deo koda aplikacije, i ostaje aktivan sve dok više nije potreban i dok sistemu ne bude potrebno da mu oduzme memoriju kako bi je koristile druge aplikacije. To je jedna od neobičnih ali fundamentalnih osobina Androida – životni ciklus procesa aplikacije ne zavisi od same aplikacije, odnosno sama aplikacija ne može direktno kontrolisati svoj proces. Životni ciklus procesa zavisi od sistema, koji o tome odlučuje na osnovu sledećih nekoliko stavki:

- za koje komponente aplikacije sistem zna da se izvršavaju,
- koliko su te komponente bitne za korisnika, i
- koliko je memorije još dostupno sistemu.

Programeri bi trebalo da razumeju koliko različite komponente aplikacije (aktivnosti, servisi, broadcast receiveri) utiču na životni ciklus samog procesa aplikacije. Ukoliko se komponente ne koriste na adekvatan način, može doći do situacije da sistem ubije proces aplikacije dok ona izvršava neki važan posao.

Odluku o tome koji proces treba ubiti kada sistemu nedostaje memorija Android donosi na osnovu nekoliko pravila. Android najpre pravi hijerarhiju u koju ređa procese na osnovu njihove važnosti (slika 5.1). Važnost procesa se određuje na osnovu komponenti koje se u njemu izvršavaju i njihovog stanja.



Slika 5.1, hijerarhija procesa po važnosti

Najviši prioritet imaju aktivni procesi (engl. *foreground process*). To su procesi aplikacije koja je u prvom planu i čije komponente trenutno vrše interakciju sa korisnikom, ili rade nešto što je korisniku trenutno potrebno. Proces se smatra da je u prvom planu ukoliko je ispunjen bilo koji od sledećih uslova:

- Aplikacija ima aktivnost koja je u prvom planu i sa kojom korisnik trenutno vrši interakciju (nakon poziva njene onResume() metode).
- Aplikacija ima BroadcastReceiver koji se trenutno izvršava (odnosno izvršava se njegova metoda onReceive()).
- Aplikacija ima servis koji trenutno izvršava kod u jednoj od svojih callback metoda (onCreate(), onStart(), onDestroy()).

U svakom trenutku ima svega nekoliko procesa u celom sistemu koji su u aktivnom stanju, i oni će biti ubijeni samo u krajnjem slučaju da je raspoloživa memorija toliko niska da ne može da održi ni ove procese da se ispravno izvršavaju.

Drugi nivo prioriteta su vidljivi procesi. Vidljiv proces izvršava neki posao i korisnik je svestan toga, pa bi ubijanje ovakvog procesa bilo primetno od strane korisnika i imalo bi negativan uticaj na korisničko iskustvo. Proces se smatra vidljivim u sledećim uslovima:

- Aplikacija izvršava aktivnost koja je vidljiva korisniku na ekranu, ali nije u prvom planu (pozvana je njena onPause() metoda). Tipičan primer je ukoliko je aktivnost koja je u prvom planu prikazana kao dijalog koji ne zauzima ceo ekran, već se vidi i deo prethodne aktivnosti

iza nje.

- Aplikacija ima servis koji se izvršava u prvom planu.

- Aplikacija ima servis koju sistem koristi za određenu funkcionalnost koje je korisnik svestan, poput live pozadine i slično.

Ovakvi procesi se smatraju za ekstremno važne i sistem ih neće ubiti sve dok ima dovoljno memorije da izvršava sve procese koji su u prvom planu.

Treći nivo prioriteta su servisni procesi, odnosno procesi koji imaju pokrenut servis pozvan sa startService() metodom. Iako ovi procesi nisu direktno vidljivi korisniku, u opštem slučaju rade stvari koje su važne korisniku (preuzimanje podataka preko mreže u pozadini na primer), pa će se sistem truditi da održi ove procese u životu sve dok ima dovoljno memorije da se podrže svi aktivni procesi koji su u prvom planu i svi vidljivi procesi.

Servisima koji se izvršavaju dugo (30 minuta i više) može biti smanjen prioritet kako bi se dozvolilo da njihov proces padne u LRU listi (engl. *Least Recently Used*). Na taj način se izbegava situacija gde dugotrajni servisi sa eventualnim curenjem memorije mogu da zauzmu preveliku količinu RAM-a i ugroze performanse sistema.

Četvrti nivo prioriteta su pozadinski procesi. Pozadinski, odnosno keširan proces je takav proces koji trenutno nije potreban, pa sistem može da ga ubije po želji ukoliko je memorija potrebna drugim procesima. Ovakvi procesi tipično imaju aktivnosti koje nisu trenutno vidljive korisniku (pozvana metoda onStop()). Ukoliko je aktivnost dobro programirana sa ispravnim ponašanjem onStop() metode, ubijanje ovakvog procesa neće uticati na korisnika kada se vrati u aplikaciju, jer će aktivnost prilikom ponovnog pokretanja moći da restaurira svoje stanje. U normalno održavanom sistemu, samo ovi procesi ulaze u razmatranje za ubijanje. Dobro održavan sistem ima uvek nekoliko dostupnih keširanih procesa (kako bi se obezbedilo efikasno prelaženje između aplikacija), a redovno će ubijati najstarije keširane procese. Samo u slučaju kritičnih situacija sistem može doći do tačke da su svi keširani procesi ubijeni, pri čemu i dalje nema dovoljno memorije pa mora početi da ubija i servisne procese. Ovakvi procesi se drže u LRU listi, gde će poslednji proces u listi biti prvi ubijen kako bi se oslobodila memorija.

Najniži nivo prioriteta imaju prazni procesi – to su aplikacije bez i jedne aktivne komponente, na samom kraju svog životnog ciklusa.

Sada, kada je poznat životni ciklus same aplikacije, može se detaljno obratiti pažnja na životni ciklus pojedinačnih aktivnosti, koji je opisan u nastavku.

5.3. Životni ciklus aktivnosti

Tokom svog životnog ciklusa, svaka aktivnost prolazi kroz određeni broj stanja. Korisnik aplikacije može da se kreće kroz aplikaciju, da izađe iz nje, da se ponovo vrati i slično. Za to vreme, aktivnosti aplikacije prolaze kroz različita stanja u svom životnom ciklusu. Activity klasa nudi nekoliko callback metoda koje omogućavaju aktivnosti da zna da joj se stanje promenilo i da po potrebi reaguje na tranziciju. Promene od značaja su kada sistem pokreće aktivnost, stopira aktivnost, nastavlja aktivnost, ili uništava proces kome aktivnost pripada.

Programer može da se „zakači“ na callback metode koje aktivnost pruža i definiše kako će se aktivnost ponašati u slučajevima kada korisnik aplikacije izađe iz aktivnosti ili ponovo uđe u nju. Na primer, ukoliko aktivnost strimuje video sa Interneta, trebalo bi pauzirati video i isključiti mrežnu konekciju ukoliko se korisnik prebaci na drugu aplikaciju. Kada se korisnik vrati u aplikaciju, potrebno je ponovo se povezati na mrežu i nastaviti video sa mesta gde je pauziran. Slično, ukoliko aplikacija prikazuje interaktivni grafički sadržaj (poput mobilnih igara), nema smisla ažurirati grafički displej ukoliko se korisnik prebacio na drugu aplikaciju. Kada se korisnik vrati, potrebno je ponovo početi sa ažuriranjem interfejsa. Svaka callback metoda dozvoljava da se izvrši akcija koja je odgovarajuća datoj promeni stanja. Dobra aplikacija se svodi na to da je potrebno uraditi pravu akciju u pravo vreme kako bi se tranzicije između stanja izvršile korektno, čime i aplikacija postaje robusnija i generalno poboljšava svoje performanse. Ukoliko tranzicije nisu realizovane korektno, aplikacija se može susresti sa čitavim nizom problema, čime se drastično smanjuje njena upotrebljivost i snižava zadovoljstvo korisnika:

- Nasilni prekid rada aplikacije (engl. *crash*) ukoliko korisnik tokom korišćenja aplikacije primi telefonski poziv ili se prebaci na drugu aplikaciju.
- Nerazumno trošenje vrednih i veoma ograničenih sistemskih resursa u trenutku dok korisnik ne koristi aktivno aplikaciju.
- Gubitak korisničkih podataka ukoliko korisnik napusti aplikaciju i kasnije je ponovo otvori.
- Nasilni prekid rada aplikacije ili gubitak korisničkih podataka ukoliko se ekran zarotira.

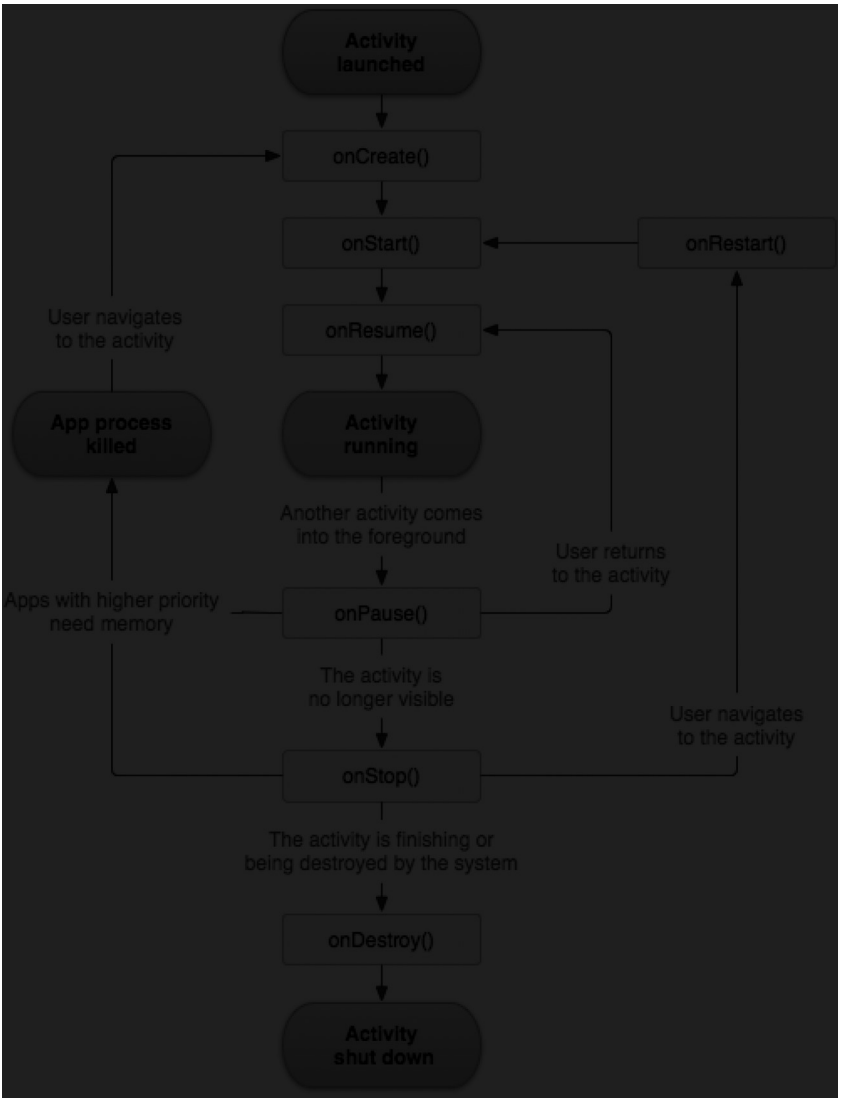
Klasa Activity nudi šest callback metoda kako bi omogućila korektnu tranziciju između stanja životnog ciklusa aktivnosti. Ove metode poziva Android sistem kada aktivnost pređe u novo stanje, i to su redom:

- onCreate(),
- onStart(),

- onResume(),

- onPause(),
- onStop(), i
- onDestroy.

Ovaj koncept prikazan je na slici 5.2, na kojoj su označena stanja i callback metode (izvor: https://developer.android.com/guide/components/activities/activity-lifecycle).



Slika 5.2, životni ciklus aktivnosti sa callback metodama

U zavisnosti od aplikacije koja se razvija i od same složenosti aktivnosti zavisi koje callback metode je potrebno implementirati. Najčešće nije potrebno implementirati svih šest metoda za svaku aktivnost, ali potrebno je razumeti ponašanje svake od ovih metoda i implementirati odgovarajuće kako bi se aktivnost ponašala onako kako korisnik aplikacije i očekuje. Dodatno, ove callback metode ne poziva programer, već ih sve okida Android sistem prilikom tranzicije između različitih stanja u kojima se aktivnost može naći.

Ključni aspekt koji je potrebno razumeti jeste trenutak kada korisnik počne da napušta aktivnost (na primer, otvori novu aplikaciju). U tom trenutku Android sistem počinje da poziva metode koje vrše postepenu demontažu aktivnosti. Ponekad je ova demontaža samo privremena, jer kada korisnik otvori novu aplikaciju, prethodna aktivnost i dalje postoji u memoriji, a korisnik je može lako ponovo vratiti u prvi plan. U tom slučaju aktivnost treba da nastavi tamo gde je stala pre gubitka fokusa.

onCreate()

Prva od metoda, koja je obavezni deo aktivnosti koji se mora implementirati, je onCreate() metoda. Ova metoda se okida u trenutku kada sistem kreira aktivnost. Tipični zadaci koji se obavljaju u ovoj metodi su inicijalizacija komponenti koje su potrebne aktivnosti, kreira se korisnički interfejs, povezuju podaci sa listama i slično. Time aktivnost prelazi u stanje Created. Aktivnost se ne zadržava dugo u ovom stanju. Čim se metoda onCreate() izvrši do kraja, aktivnost prelazi u stanje Started, a sistem poziva callback metode onStart() i onResume() jednu za drugom.

onStart()

Prilikom prelaska iz stanja Created u stanje Started, sistem poziva ovu metodu. Zadatak onStart() metode je da učini aktivnost vidljivom korisniku, jer se u ovom trenutku aktivnost prebacuje u prvi plan i postaje interaktivna. Tipična akcija koja se izvršava u ovoj metodi je inicijalizacija koda koji treba da opslužuje korisnički interfejs. Ova metoda se izvršava jako brzo, a aktivnost ne ostaje dugo u stanju Started. Odmah nakon završetka metode onStart() aktivnost prelazi u stanje Resumed, a sistem okida metodu onResume().

onResume()

Prilikom ulaska u stanje Resumed, aktivnost dolazi u prvi plan, a sistem okida onResume() metodu. U ovom stanju aktivnost je vidljiva i interaktivna, odnosno korisnik može da vrši različite akcije nad komponentama korisničkog interfejsa te aktivnosti. Aktivnost ostaje u ovom stanju sve dok se ne desi nešto što će oduzeti fokus aplikaciji, poput dolazećeg poziva, pokretanja druge aplikacije ili gašenja ekrana. Kada se ovakav događaj desi, aktivnost prelazi u stanje Paused, a sistem okida onPause() metodu. Ukoliko se iz stanja Paused aktivnost vrati u

stanje Resumed, sistem ponovo poziva onResume() metodu. Zbog toga, onResume() treba da inicijalizuje one komponente koje su oslobođene u okviru onPause() metode.

onPause()

Poziv ove metode je prva indikacija da korisnik napušta ovu aktivnost. To znači da aktivnost više nije u prvom planu (iako je možda još uvek vidljiva, na primer, ukoliko je korisnik uključio režim rada sa više prozora). Tipična upotreba onPause() metode uključuje pauziranje ili prilagođavanje operacija koje nije potrebno nastaviti dokle god je aktivnost pauzirana. Aktivnost može biti pauzirana iz više razloga:

- Nešto je oduzelo fokus aplikaciji, poput dolazećeg poziva, pokretanja druge aplikacije ili gašenja ekrana – ovo je i najčešći razlog.
- Od Android 7.0 (API nivo 24) pa na dalje, više aplikacija se može istovremeno izvršavati u režimu rada sa više prozora. Pošto samo jedan od ovih prozora može imati fokus u bilo kom trenutku, ostale aplikacije su pauzirane.
- Nova aktivnost je delimično prekrila ekran, poput otvorenog dijaloga. Sve dok je aktivnost vidljiva, ali nije u fokusu, ona ostaje pauzirana.

Metoda onPause() obično otpušta sistemske resurse (poput senzora ili GPS), kao i sve ono što može negativno uticati na životni vek baterije dok je aktivnost pauzirana, odnosno dok korisniku ne treba. Međutim, pošto aktivnost u pauziranom stanju može i dalje biti vidljiva delimično ili u celosti (naročito ukoliko se koristi režim rada sa više prozora), ovde ne treba otpustiti resurse koji se odnose na korisnički interfejs, već ga i dalje treba ažurirati (to treba uraditi u onStop() metodi). Dodatno, pošto je vreme izvršavanja metode onPause() veoma kratko, preporučeno je da se u ovoj metodi ne čuvaju korisnički podaci, ne prave mrežni pozivi ili izvršavaju operacije nad bazom, pošto se možda neće izvršiti do kraja pre završetka metode. Ovakve dugotrajne operacije se izvršavaju u onStop() metodi.

Aktivnost ostaje u ovom stanju i nakon završetka onPause() metode, sve dok se aktivnost ne vrati u prvi plan, ili dok ne postane potpuno nevidljiva korisniku. Ukoliko se aktivnost nastavi i vrati u prvi plan, sistem ponovo okida onResume() metodu, a aktivnost prelazi u stanje Resumed. Sa druge strane, ukoliko aktivnost postane potpuno nevidljiva, sistem poziva onStop() metodu.

onStop()

Kada aktivnost više nije vidljiva korisniku, ona prelazi u Stopped stanje, a sistem okida metodu onStop(). To je slučaj kada je nova aktivnost prekrila ceo ekran. Sistem može pozvati ovu metodu i kada je aktivnost završila sa izvršavanjem i treba da se uništi. U ovoj metodi aktivnost treba da otpusti ili

prilagodi resurse

koji nisu potrebni dok aktivnost nije vidljiva. Tipično, treba obustaviti ažuriranje korisničkog interfejsa (pošto korisnik ne vidi aktivnost, nema smisla ažurirati ekran) i izvršiti sve dugotrajne operacije poput snimanja podataka u bazu.

Aktivnost koja je stopirana se i dalje drži u memoriji, čuva svoje stanje i informacije. Ukoliko se aktivnost nastavi, svi podaci su i dalje tu. U tom slučaju, poziva se metoda onRestart(). Međutim, kada je aktivnost stopirana, sistem može uništiti proces kome pripada u bilo kom trenutku, ukoliko je potrebna memorija. Ako je aktivnost završila sa izvršavanjem, sistem poziva onDestroy() metodu.

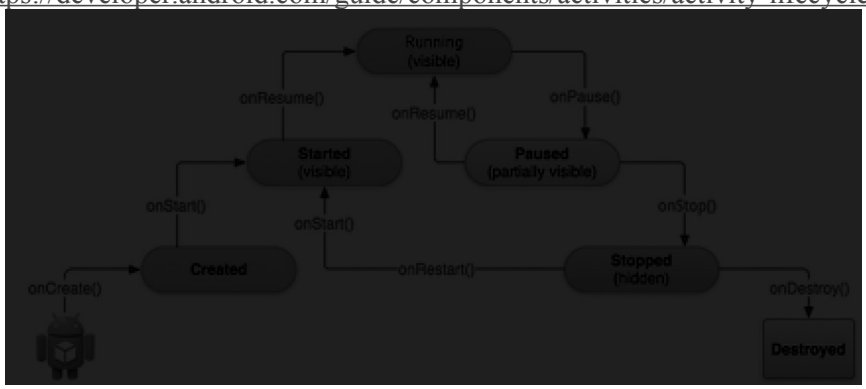
onDestroy()

Sistem poziva ovu metodu neposredno pre uništavanja aktivnosti. To se može desiti u više slučajeva:

- Aktivnost se završila (korisnik je odbacio aktivnost ili je pozvana njena finish() metoda.
- Sistem je privremeno uništio aktivnost zbog promene u konfiguraciji, poput rotacije ekrana ili ulaska u režim rada sa više prozora.

U tom trenutku aktivnost treba da počisti za sobom i otpusti sve ostale resurse pre nego što bude uništena.

Najlakši način za razumevanje životnog ciklusa aktivnosti je kroz dijagram stanja sa prikazanim metodama koje se pozivaju prilikom prelaska iz jednog stanja u drugo (slika 5.3, preuzeta sa: https://developer.android.com/guide/components/activities/activity-lifecycle).



Slika 5.3, dijagram stanja u kojima se aktivnost može naći

Kako bi se dodatno ilustrovali i demonstrirali pozivi callback metoda, neka se posmatra jednostavna aktivnost koja ima implementirane sve navedene callback metode iz životnog ciklusa aktivnosti. Log poruke se generišu pomoću Log.d()

metode, kako bi se ključni pozivi metoda ispisali u logu. Kod aktivnosti je dat na sledeći način:

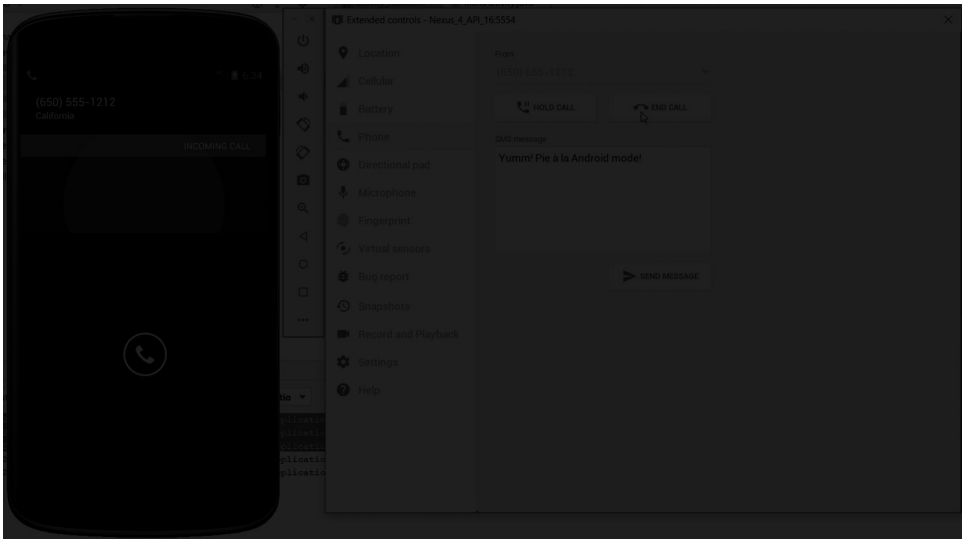
```
package com.example.student.myapplication;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

    String msg = "Android : ";

    /** Poziva se prilikom prvog kreiranja aktivnosti */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(msg, "Poziv onCreate() metode");
    }

    /** Poziva se kada aktivnost treba da postane vidljiva
*/
    @Override
    protected void onStart() {
        super.onStart();
        Log.d(msg, "Poziv onStart() metode");
    }
    /** Poziva se kada je aktivnost postala vidljiva */
    @Override
    protected void onResume() {
        super.onResume();
        Log.d(msg, "Poziv onResume() metode");
    }

    /** Poziva se kada aktivnost pocne da gubi fokus */
    @Override
    protected void onPause() {
        super.onPause();
        Log.d(msg, "Poziv onPause() metode");
    }

    /** Poziva se kada aktivnost vise nije vidljiva */
    @Override
    protected void onStop() {
        super.onStop();
        Log.d(msg, "Poziv onStop() metode");
    }
```

```
    /** Poziva se neposredno pre unistavanja aktivnosti */
    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(msg, "Poziv onDestroy() metode");
    }
}
```

Nakon pokretanja aplikacije na emulatoru i pokretanja aplikacije, u sekciji LogCat se može videti sledeći redosled poziva metoda:

```
01-25 18:22:55.122 2362-
2362/com.example.student.myapplication D/Android :: Poziv
onCreate() metode

01-25 18:22:55.132 2362-
2362/com.example.student.myapplication D/Android :: Poziv
onStart() metode

01-25 18:22:55.132 2362-
2362/com.example.student.myapplication D/Android :: Poziv
onResume() metode
```

To je i očekivan redosled, pošto je tako definisano u životnom ciklusu aktivnosti. Posmatrajmo sada situaciju da u tom trenutku, dok je posmatrana aktivnost u prvom planu, stigne telefonski poziv na uređaj, što se može simulirati upotrebom emulatora kao što je prikazano na slici 5.4.



Slika 5.4, primer aktivnosti koja gubi fokus usled dolaznog poziva

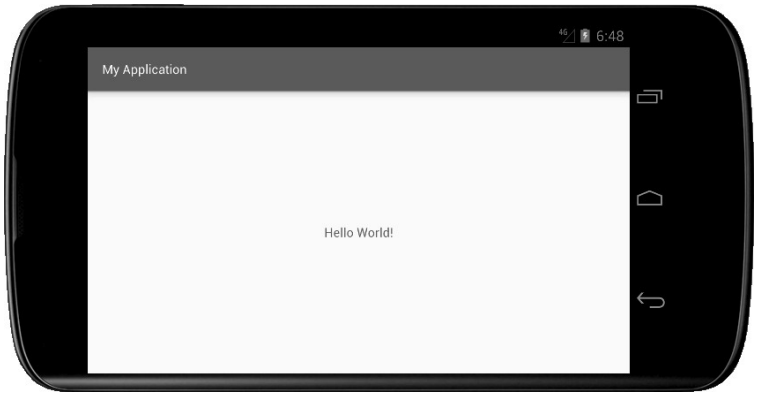
U logu se u tom trenutku mogu videti sledeća dva poziva:

```
01-25 18:33:58.672 2362-
2362/com.example.student.myapplication D/Android :: Poziv
onPause() metode

01-25 18:33:59.342 2362-
2362/com.example.student.myapplication D/Android :: Poziv
onStop() metode
```

U tom slučaju, pošto aktivnost gubi fokus jer telefonska aplikacija izlazi u prvi plan, pozivaju se prvo onPause(), a odmah nakon toga i onStop() metoda. Nakon završetka poziva, telefonska aplikacija se uklanja, a posmatrana aktivnost ponovo dolazi u prvi plan. Očekivano, nakon prekida poziva, u logu se ispisuju sledeća dva poziva (prvo poziv onStart() metode, a zatim i onResume() metode):

```
01-25 18:37:30.132 2362-
2362/com.example.student.myapplication D/Android :: Poziv
onStart() metode

01-25 18:37:30.132 2362-
2362/com.example.student.myapplication D/Android :: Poziv
onResume() metode
```

Interesantno je obratiti pažnju na to šta se dešava u slučaju da korisnik promeni orijentaciju telefona, tj. ukoliko pređe iz standardnog vertikalnog položaja (engl. *Portrait*) u horizontalni položaj (engl. *Landscape*), kao što je prikazano na slici 5.5. U tom slučaju, potrebno je znati kako se Android sistem ponaša. Aktivnost koja se trenutno izvršava se restartuje, odnosno uništava i ponovo kreira (u logu će se videti novi poziv onCreate() metode). Ovakvo ponašanje je očekivano i implementirano namerno, kako bi se pomoglo aplikaciji da se adaptira na novu konfiguraciju ekrana automatskim restartom. Međutim, ukoliko programer ne očekuje ovakvo ponašanje, može doći do gubitka nekih korisničkih podataka koje sadrži aktivnost. Poželjno je koristiti neki oblik perzistencije kako bi se očuvalo stanje aktivnosti.



Slika 5.5, uticaj promene rotacije uređaja na stanje aktivnosti

Ukoliko se zarotira ekran, u logu su ispisani sledeći pozivi:

```
01-25 19:24:07.882 2676-
2676/com.example.student.myapplication D/Android :: Poziv
onPause() metode

01-25 19:24:07.882 2676-
2676/com.example.student.myapplication D/Android :: Poziv
onStop() metode

01-25 19:24:07.882 2676-
2676/com.example.student.myapplication D/Android :: Poziv
onDestroy() metode

01-25 19:24:07.942 2676-
2676/com.example.student.myapplication D/Android :: Poziv
onCreate() metode

01-25 19:24:07.942 2676-
2676/com.example.student.myapplication D/Android :: Poziv
onStart() metode

01-25 19:24:07.942 2676-
2676/com.example.student.myapplication D/Android :: Poziv
onResume() metode
```

Ukoliko korisnik izađe iz aktivnosti pritiskom na dugme Back, aktivnost se uništava, što se vidi u sledećim pozivima koji su ispisani u logu:

```
01-25 19:27:47.542 2676-
2676/com.example.student.myapplication D/Android :: Poziv
onPause() metode

01-25 19:27:48.102 2676-
2676/com.example.student.myapplication D/Android :: Poziv
onStop() metode

01-25 19:27:48.102 2676-
2676/com.example.student.myapplication D/Android :: Poziv
onDestroy() metode
```

5.4. Čuvanje stanja korisničkog interfejsa

U trenutku kada aktivnost počinje da prelazi u stopirano stanje, sistem automatski poziva metodu onSaveInstanceState(), čime se aktivnosti pruža šansa da sačuva informacije o svom stanju. To je neophodno jer ne postoje garancije da će aktivnost koja je stopirana ponovo biti pozvana u prvi plan, već postoje dobre šanse da će je Android sistem nakon određenog vremena uništiti. Podrazumevana implementacija ove metode čuva privremene podatke o

stanju komponenti u

hijerarhiji ekrana, poput informacije o tekstu unetom u EditText komponentu, ili poziciju klizača (scroll) u ListView rasporedu. Ukoliko je potrebno sačuvati bilo koje dodatne podatke o stanju aktivnosti, mora se nadjačati onSaveInstanceState(). U nadjačanoj metodi treba dodati parove ključ-vrednost u Bundle objekat koji se čuva u slučaju da se aktivnost neočekivano uništi. Bundle objekti se često upotrebljavaju u Android programiranju, tipično za prenos podataka između različitih aktivnosti. Po implementaciji Bundle veoma liči na Java Map objekat, koji mapira String ključeve u vrednosti. Razlog zašto Android ne koristi stare Map objekte je da su mape previše fleksibilne i mogu da sadrže objekte koji ne mogu biti serijalizovani. Bundle ograničava tipove objekata koji se mogu dodati u Bundle objekat tako da je sadržaj Bundle objekta garantovano serijalizibilan.

Prilikom nadjačavanja metode onSaveInstaceState(), mora se pozvati verzija iz bazne klase ukoliko se želi i podrazumevano ponašanje, odnosno čuvanje stanja hijerarhije komponenti. Dodatna napomena se odnosi na sam Bundle objekat, koji je prihvatljivo rešenje samo u slučaju da je potrebno sačuvati veoma mali skup podataka (zbog toga što zahteva serijalizaciju i izvršava se u glavnoj niti aplikacije). Za čuvanje podataka većeg obima postoje drugi oblici perzistencije, o kojima će biti reči kasnije.

Neka se kao primer posmatra aktivnost neke mobilne igre, u kojoj je neophodno sačuvati i podatke o trenutno ostvarenim poenima igrača i o njegovom nivou. U tom slučaju, potrebno je nadjačati onSaveInstanceState() metodu i u njoj sačuvati potrebne niformacije u Bundle objektu na sledeći način:

```
static final String STANJE_POENI = "stanjePoeni";
static final String STANJE_NIVO = "stanjeNivo";

// ...

@Override
public void onSaveInstanceState(Bundle savedInstanceState)
{
    // Sacuvati trenutno stanje igre
    savedInstanceState.putInt(STANJE_POENI, trenutniPoeni);
    savedInstanceState.putInt(STANJE_NIVO, trenutniNivo);
    // Pozvati verziju iz bazne klase da sacuva hijerarhiju
komponenti
    super.onSaveInstanceState(savedInstanceState);
}
```

Prilikom pokretanja aktivnosti nakon što je prethodno uništena, njeno stanje se može restaurirati iz sačuvanog Bundle objekta koje će sistem proslediti aktivnosti. Dve callback metode prihvataju isti Bundle objekat koji sadrži informacije o prethodnom stanju:

- onCreate(),
- onRestoreInstanceState().

Pošto se onCreate() metoda poziva i u slučaju da sistem kreira novu instancu aktivnosti i u slučaju da restaurira prethodno uništenu, pre pokušaja čitanja potrebno je prvo proveriti da li je Bundle objekat null. Ukoliko je null, to znači da sistem kreira novu instancu aktivnosti, i da nema šta da se restaurira.

Ako se posmatra prethodni primer mobilne igre gde se čuvaju trenutno ostvareni poeni i nivo korisnika, implementacija u slučaju da se restauracija radi kroz onCreate() metodu data je sledećim isečkom Java koda:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Uvek pozvati prvo
metodu iz bazne klase

    //  Proveriti  da  li  restauriramo  prethodno
        unistenu instancu ili pravimo potpuno novu instancu
    if (savedInstanceState != null) {
        // Restaurirati sacuvane vrednosti iz Bundle
        objekta trenutniPoeni =
savedInstanceState.getInt(STANJE_POENI);
        trenutniNivo =
savedInstanceState.getInt(STANJE_NIVO);
    } else {
        //   Kreira   se   nova   instanca,   potrebno
             je inicijalizovati promenljive na pocetne
vrednosti...
    }
    // ...
```

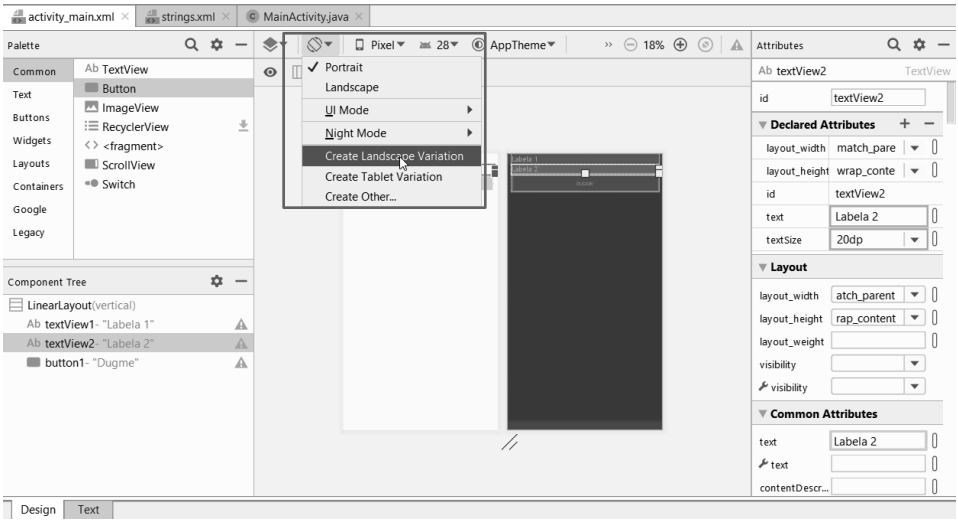
Umesto restauracije tokom metode onCreate(), može se implementirati callback onRestoreInstanceState() metoda, koju Android sistem poziva nakon onStart() metode samo u slučaju da postoji sačuvano stanje koje treba restaurirati. Samim tim, nije potrebno proveravati da li je Bundle objekat null. Ovakva implementacija data je sledećim isečkom Java koda:

```
public void onRestoreInstanceState(Bundle
savedInstanceState) {
    // Prvo pozvati verziju iz bazne klase da
restaurira hijerarhiju komponenti
    super.onRestoreInstanceState(savedInstanceState);
    // Restaurirati sacuvane vrednosti iz Bundle
    objekta trenutniPoeni =
savedInstanceState.getInt(STANJE_POENI);
    trenutniNivo = savedInstanceState.getInt(STANJE_NIVO);
}
```

5.5. Promena orijentacije ekrana

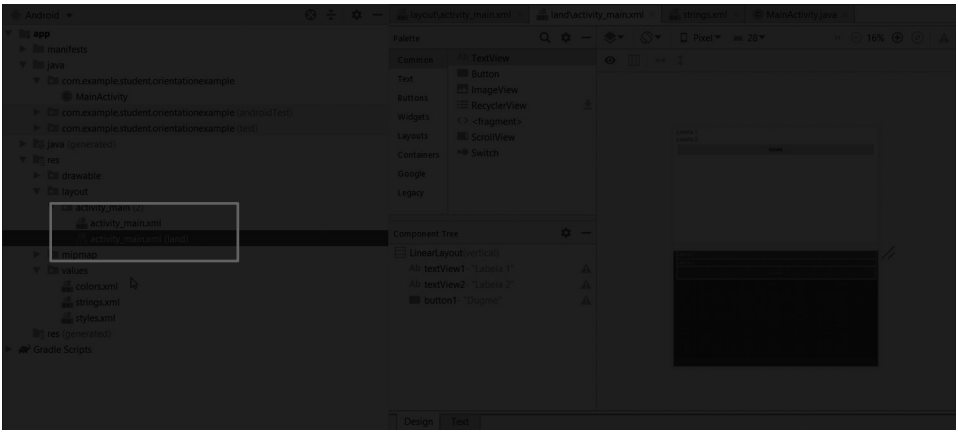
Konfiguracija uređaja se može promeniti u toku izvršavanja aplikacije, a tipični ovakvi primeri su promena orijentacije uređaja ili ulazak u režim rada sa više prozora. U slučaju da se ovakva situacija desi, Android restartuje aktivnost koja se trenutno izvršava tako što poziva prvo onDestroy(), a zatim i onCreate() metode. Ovakvo ponašanje je dizajnirano sa ciljem da se aplikaciji pomogne da se adaptira na novu konfiguraciju i automatski ponovo učita alternativne resurse koji odgovaraju novoj konfiguraciji. Kada se posmatra orijentacija ekrana, postoje Portrait orijentacija (standardni prikaz, kada se telefon drži vertikalno) i Landscape orijentacija (telefon u vodoravnom položaju).

Android sistem će prilikom promene orijentacije uređaja iz Portrait u Landscape režim sam iscrtati korisnički interfejs po pravilima koja su zadata u XML za podrazumevani (Portrait) režim. Ponekad to nije ono što programer želi, već je potrebno iscrtati drugačiji raspored komponenti. Alternativni resurs za Portrait režim se dodaje jednostavno u vizuelnom editoru odabirom opcije Create Landscape Variation, kao što je prikazano na slici 5.6.



Slika 5.6, dodavanje alternativnog prikaza za Landscape režim

Nakon dodavanja alternativnog prikaza za Landscape režim, biće dodat još jedan XML fajl sa rasporedom komponenti u okviru res/layout foldera, koji će imati isto ime kao i podrazumevani fajl za Portrait režim, uz oznaku (land), kao što je prikazano na slici 5.7.



Slika 5.7, dodatni XML fajl za raspored komponenti u Landscape režimu

Neka se posmatra sledeći primer, gde raspored komponenti u Portrait režimu koristi linearni vertikalni raspored za prikaz komponenti. Neka je XML kod fajla activity_main.xml dat sa:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="10dp"
    android:paddingRight="10dp"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Labela 1"
        android:textSize="20dp" />
    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Labela 2"
        android:textSize="20dp" />
    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Dugme" />
</LinearLayout>
```

Pri prelasku u Landscape režim, ne želimo da se koristi isti raspored komponenti, već je potrebno da se koristi linearni horizontalni raspored. U tom slučaju, dodaje se još jedan XML fajl koji odgovara Landscape orijentaciji uređaja (activity_main.xml (land)), sa istim komponentama, ali organizovanim u drugačijem rasporedu, i njegov sadržaj dat je u nastavku:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="10dp"
    android:paddingRight="10dp"
    android:orientation="horizontal" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Labela 1"
        android:textSize="20dp" />
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Labela 2"
        android:textSize="20dp" />
    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Dugme" />
</LinearLayout>
```

Kada su XML fajlovi ovako definisani, Android sistem će prilikom rotacije uređaja restartovati aktivnost, koja će nakon toga tokom novog izvršavanja svoje onCreate() metode učitati Landscape XML fajl. Prikaz korisničkog interfejsa definisanog na ovaj način, u oba režima rada, prikazan je na slici 5.8.

Slika 5.8, korisnički interfejs aplikacije sa odvojenim rasporedima za Portrait i Landscape režim

Na promenu konfiguracije uređaja moguće je reagovati i iz Java koda odgovarajuće aktivnosti. Prilikom promene konfiguracije, Android sistem generiše događaj koji je moguće uhvatiti callback metodom onConfigurationChange u kodu aktivnosti i samostalno reagovati na promenu koja se desila. Ovoj metodi se kao parametar prosleđuje objekat tipa Configuration koji specificira novu konfiguraciju uređaja. Čitanjem odgovarajućih polja iz Configuration objekta, moguće je odrediti šta je promenjeno i reagovati ažuriranjem resursa koji se koriste u korisničkom interfejsu. Ipak, treba biti oprezan jer samostalno rukovanje promenama u konfiguraciji može drastično zakomplikovati i otežati upotrebu alternativnih resursa, zato što ih sistem u tom slučaju neće automatski primeniti umesto programera. Ova tehnika nije preporučena od strane Android zajednice.

Ukoliko je ipak potrebno samostalno rukovati promenama orijentacije uređaja u aktivnosti, mora se u manifestu aplikacije za tu aktivnost dodati atribut android:configChanges, i deklarisati sledeće vrednosti: "orientation", "screenSize" i "screenLayout". Na primer, sadržaj manifesta u slučaju da aktivnost treba da reaguje na promenu konfiguracije i u slučaju promene orijentacije, kao i u slučaju promene statusa dostupnosti tastature ("keyboardHidden" vrednost) bi bio sledeći:

```
<activity android:name=".MainActivity"
          android:configChanges= "orientation|screenSize|
screenLayout|keyboardHidden"
          android:label="@string/app_name">
```

U ovom slučaju, ukoliko se bilo koja od ovih promena konfiguracije desi, MainActivity se neće automatski restartovati, već će biti pozvana callback metoda onConfigurationChanged() u okviru MainActivity klase. Primer hvatanja ovog poziva u telu MainActivity, koji nakon toga proverava trenutnu orijentaciju uređaja dat je u nastavku:

```
@Override
public void onConfigurationChanged(Configuration newConfig)
{
    super.onConfigurationChanged(newConfig);

    // Provera orijentacije ekrana
    if (newConfig.orientation ==
Configuration.ORIENTATION_LANDSCAPE) {
        Toast.makeText(this, "landscape",
Toast.LENGTH_SHORT).show();
    } else if (newConfig.orientation ==
Configuration.ORIENTATION_PORTRAIT){
        Toast.makeText(this, "portrait",
Toast.LENGTH_SHORT).show();
    }
}
```

5.6. Intenti (namere)

Skoro svaka Android aplikacija se sastoji od više aktivnosti. Tokom rada aplikacije, vrlo je verovatno da će korisnik da ulazi i izlazi iz jedne aktivnosti više puta, kao i da će biti potrebno da se iz jedne aktivnosti pokrene neka druga. To je slučaj kada aplikacija treba da pređe sa ekrana na kome se trenutno korisnik nalazi na novi ekran. Pokretanje nove aktivnosti se može postići pozivom jedne od dve moguće metode:

- startActivity(), ukoliko nova aktivnost ne treba da vrati rezultat,
- startActivityForResult(), ukoliko nova aktivnost treba da vrati rezultat.

Obe metode prihvataju kao argument objekat tipa Intent (engl. namera). Ovaj objekat definiše ili tačno određenu aktivnost koja treba da se pokrene, ili tip akcije koja se želi izvršiti (pri čemu će sistem odrediti odgovarajuću aktivnost, koja može pripadati i nekoj drugoj aplikaciji). Osim toga, Intent objekat može da se iskoristi da se novoj aktivnosti prosledi manja količina podataka koji su joj potrebni.

Kada jedna aktivnost pokrene neku drugu aktivnost, obe će proći kroz promene stanja. Prva aktivnost prestaje sa radom i prelazi u pauzirano ili stopirano stanje, dok se druga aktivnost kreira. Redosled poziva callback metoda u svakoj

aktivnosti je jasno definisan, naročito u slučaju da su aktivnosti deo iste aplikacije. U slučaju da aktivnost A pokreće aktivnost B, redosled je sledeći:

- Aktivnost A izvršava onPause(),
- Aktivnost B izvršava redom onCreate(), onStart() i onResume() i dobija fokus,
- Ukoliko aktivnost A više nije vidljiva na ekranu, izvršava onStop() metodu.

Jasno definisan i poznat redosled poziva značajno olakšava implementaciju prelaska iz jedne aktivnosti u drugu.

5.6.1. Intent objekat

Intent objekat služi kao apstraktni opis neke operacije koja treba da se izvrši. Može da koristi za:

- pokretanje nove aktivnosti (poziv startActivity(Intent)),
- komunikaciju za pozadinskim servisom (poziv startService(Intent)),
- komunikaciju sa BroadcastReceiver komponentom (broadcastIntent()).

Osim što se objekat klase Intent koristi za pokretanje nove aktivnosti, može da posluži i za prenos informacija između različitih komponenti sistema. Informacije koje mogu da budu prenete su definisane atributima klase Intent:

- ime komponente (component) – opcioni parametar, koji se koristi prilikom pokretanja eksplicitne namere. Definiše ime komponente kojoj se prosleđuje ovaj objekat. Ukoliko ovaj parametar nije unet, radi se o implicitnoj nameri, odnosno Android sistem treba na osnovu vrednosti ostalih atributa da odredi kome treba da isporuči ovaj objekat.
- akcija (action)– uopšten opis akcije koju je potrebno obaviti. Klasa Intent ima veći broj konstanti koji opisuje generičke akcije, poput ACTION_SEND (koristi se kada je potrebno deliti podatke kroz neku drugu aplikaciju) ili ACTION_VIEW (uopšten opis akcije kada je potrebno prikazati neke podatke korisniku). Moguće je definisati i svoje akcije.
- podaci (data i/ili type) – koristi se za prosleđivanje podataka kroz URI objekat koji može da ukazuje na same podatke i na njihov MIME tip.
- kategorija (category) - dodatne informacije o akciji koju treba izvršiti. Na primer CATEGORY_LAUNCHER označava da je aktivnost inicijalna aktivnost nekog zadatka i da se nalazi u listi aplikacija koje izlistava sistem, dok CATEGORY_BROWSABLE dozvoljava da

aktivnost bude pokrenuta kroz veb čitač kako bi se prikazali podaci na koje ukazuje data adresa.

- extras – Bundle objekat koji sadrži sve dodatne informacije u obliku parova ključ vrednost. Na primer, prilikom kreiranja Intent objekta za slanje mejla sa akcijom ACTION_SEND, moguće je specificirati primaoca sa ključem EXTRA_EMAIL, a naslov mejla ključem EXTRA_SUBJECT. U samoj klasi Intent postoji veći broj konstanti u obliku EXTRA_ za standardne tipove informacija koje se prenose. Naravno, moguće je i samostalno definisati dodatne ključeve.

5.6.2. Eksplicitne i implicitne namere

Ukoliko nova aktivnost ne treba da vrati nikakav rezultat svog izvršavanja, može se pokrenuti prostim pozivom metode startActivity(). Ukoliko je potrebno iz MainActivity jednostavno pokrenuti aktivnost DrugaActivity, to se može postići sledećim delom Java koda:

```
Intent intent = new Intent(this, DrugaActivity.class);
startActivity(intent);
```

Ovakav tip poziva je poznat kao **eksplicitna namera** – tačno je navedeno koja aktivnost treba da se pokrene. Sa druge strane, ponekad je samo potrebno izvršiti neku akciju koja je opisno zadata, poput slanja mejla ili pravljenja fotografije. Naša aplikacija čak i ne mora imati aktivnosti koje mogu da izvrše takve akcije, pa se može od samog Android sistema zahtevati da pronađe neku aktivnost u okviru neke druge aplikacije na uređaju koja može uraditi traženu akciju. Ovakav tip namere poznat je kao **implicitna namera**, gde se samo opiše akcija koju je potrebno izvršiti, bez tačnog navođenja koju konkretnu aktivnost treba pozvati.

Kako bismo ilustrovali implicitnu nameru, neka se posmatra slučaj da je aplikaciji potrebna usluga kamere, pošto je potrebno da se iz aplikacije može napraviti fotografija. Jedna od mogućih opcija bila bi da se u potpunosti od nule razvije aktivnost u kojoj će biti implementirana kompletna funkcionalnost kamere. Međutim, ovakav zadatak nije nimalo jednostavan iz dva razloga:

- kod koji bi trebalo razviti je složen,
- na samom telefonu već postoji dostupna aplikacija kamera, koja radi tačno ono što nam je potrebno, pa bi razvoj naše verzije bilo izmišljanje tople vode.

Dakle, umesto da sami razvijamo aktivnost koja bi obavljala kompletnu funkcionalnost kamere, možemo u objektu tipa Intent opisno navesti da nam je potrebna akcija snimanja fotografije i dozvoliti Android sistemu da pronađe

odgovarajuću aktivnost za nas. Konkretno, akcija koju je potrebno navesti u

ovom slučaju je MediaStore.ACTION_IMAGE_CAPTURE. U ovom slučaju, potrebno je da se dobije rezultat kada se pozvana aktivnost završi – konkretno, napravljena fotografija. Običan poziv startActivity() nije dovoljan, već se mora koristiti poziv startActivityForResult(Intent, int), koji osim samog objekta tipa Intent prihvata kao parametar i celobrojnu vrednost koja služi kao identifikator poziva. Ovaj identifikator je neophodan u slučaju da je potrebno razlikovati više različitih poziva startActivityForResult() iz iste aktivnosti. Rezultat će se vratiti prvoj aktivnosti kroz callback metodu onActivityResult(int, int, Intent). Prva celobrojna vrednost odgovara identifikatoru poziva pod kojim je aktivnost pozvana. Drugi parametar je kod rezultata, koji najčešće ima standardne vrednosti RESULT_CANCELED i RESULT_OK. Poslednji parametar je Intent objekat koji može u sebi sadržati dodatne podatke (u našem primeru to je napravljena fotografija). Kompletan primer koda sa implicitnim pozivom kamere dat je u nastavku.

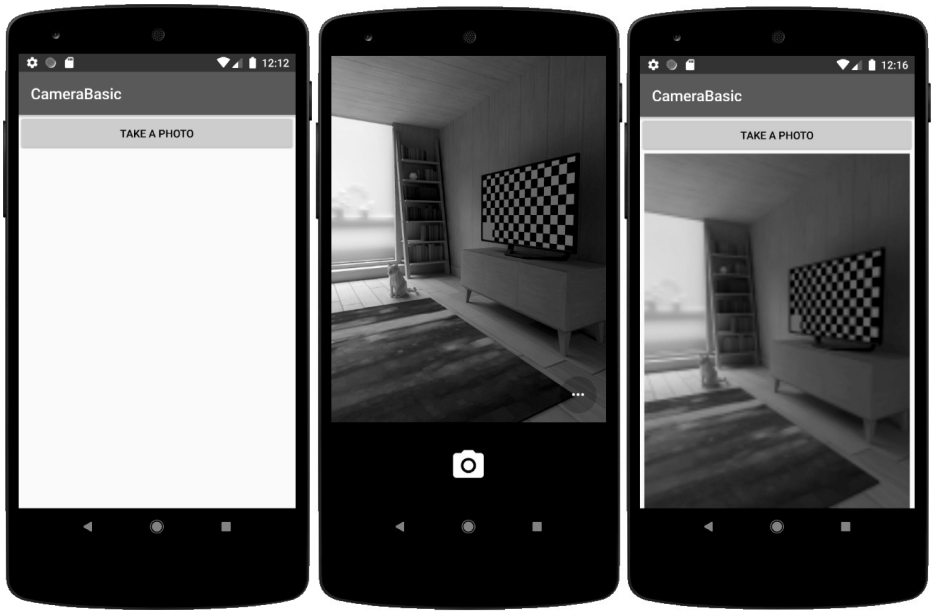
```
package com.example.student.camerabasic;

import android.content.Intent;
import android.graphics.Bitmap;
import android.provider.MediaStore;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    private Button btnCapture;
    private ImageView imgCapture;
    private static final int IMAGE_CAPTURE_CODE = 1;

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btnCapture = findViewById(R.id.btnTakePicture);
        imgCapture = findViewById(R.id.capturedImage);
        btnCapture.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent cInt = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);
```

```
startActivityForResult(cInt,IMAGE_CAPTURE_CODE);
            }
        });
    }
    @Override
    protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
        if (requestCode == IMAGE_CAPTURE_CODE) {
            if (resultCode == RESULT_OK) {
                Bitmap bp = (Bitmap)
data.getExtras().get("data");
                imgCapture.setImageBitmap(bp);
            } else if (resultCode == RESULT_CANCELED) {
                Toast.makeText(this, "Cancelled",
Toast.LENGTH_LONG).show();
            }
        }
   }
```

Dati kod podrazumeva da je u XML fajlu sa rasporedom komponenti definisano jedno dugme sa Id btnTakePicture, na čiji klik će se okinuti implicitni poziv aktivnosti kamere, kao i polje za prikaz uslikane fotografije koja će biti vraćena kao rezultat, sa Id capturedPicture. Kod activity_main.xml fajla dat je u nastavku:

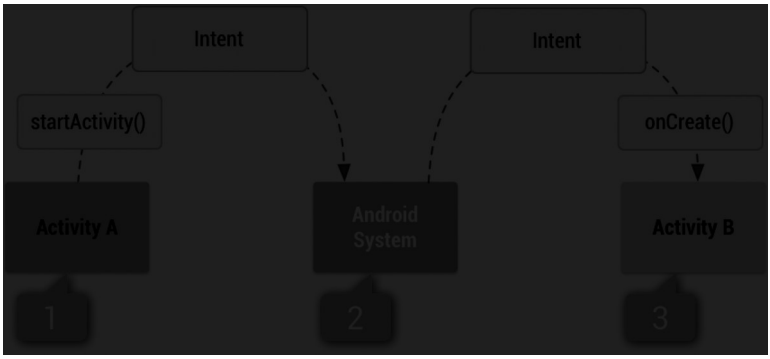
```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:id="@+id/btnTakePicture"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Take a photo" />
    <ImageView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@+id/capturedImage"/>
</LinearLayout>
```

Prikaz rada ove jednostavne aplikacije dat je na slici 5.9. Na prvom delu slike se vidi prikaz implementirane aktivnosti, na drugom delu slike prikazana je pokrenuta aktivnost kamere nakon pritiska na dugme Take a photo, a na trećem delu slike prikazana je vraćena fotografija u prvoj aktivnosti.



Slika 5.9, implicitna namera i poziv aktivnosti kamere

Na slici 5.10 prikazano je kako se koristi Intent objekat prilikom pokretanja nove aktivnosti. U slučaju eksplicitne namere, sistem odmah pokreće navedenu aktivnost. U slučaju implicitne namere, Android sistem pronalazi odgovarajuću komponentu poređenjem sadržaja Intent objekata sa Intent filterima deklarisanim u manifest fajlovima drugih aplikacija na uređaju. Ukoliko se pronađe poklapanje, sistem pokreće tu komponentu i prosleđuje joj Intent objekat. U slučaju da je pronađeno više kompatibilnih komponenti, sistem prikazuje dijalog u kome korisnik može odabrati aplikaciju koju želi da koristi.



Slika 5.10, prosleđivanje implicitne namere
(izvor: https://developer.android.com/guide/components/intents-filters#Types)

Ukoliko više aplikacija može da odgovori jednoj implicitnoj nameri, korisnik će možda želeti svaki put da odabere drugačiju aplikaciju, pa mu se može eksplicitno prikazati dijalog za odabir. Na primer, ukoliko se iz aplikacije uradi tzv. share upotrebom akcije ACTION_SEND, korisnik će možda u zavisnosti od situacije želeti da odabere drugačiju aplikaciju. Prikazani dijalog će izlistati sve aplikacije koje mogu da odgovore na zahtevanu akciju. Primer koda za prikazivanje dijaloga za odabir dat je u nastavku, a izgled ovog dijaloga

```
Intent sendIntent = new Intent(Intent.ACTION_SEND);
...

// Uvek koristiti string resurse za UI text.
// Ovaj resurs moze sadrzati na primer "Share this
photo with"
String title =
getResources().getString(R.string.chooser_title);
// Kreiranje intenta za prikaz chooser dialoga
Intent chooser = Intent.createChooser(sendIntent, title);

// Verifikacija da se originalni intent razresi na
bar jednu aktivnost
if (sendIntent.resolveActivity(getPackageManager()) !
= null) {
    startActivity(chooser);
}
```

prikazan je na slici 5.11.

Slika 5.11, dijalog za odabir aplikacije
(izvor: https://developer.android.com/guide/components/intents-

filters#ForceChooser)

5.6.3. Intent filteri i rezolucija Intenta

Ukoliko je potrebno proglasiti da aplikacija treba da prihvati implicitne namere, potrebno je definisati jedan ili više Intent filtera za svaku pojedinačnu komponentu aplikacije. To se postiže dodavanjem <intent-filter> elemenata ugnježdenog u element odgovarajuće komponente (na primer <activity>) u okviru manifest fajla. Svaki pojedinačni Intent filter definiše tip namera koje prihvata na osnovu akcije, podataka i kategorije primljenog Intent objekta. Android sistem prosleđuje implicitnu nameru aplikaciji samo u slučaju da Intent objekat prolazi jedan od filtera. Na primer, ukoliko je potrebno deklarisati da aktivnost može da prihvati Intent sa akcijom ACTION_SEND pri čemu je tip podatka text, u okviru elementa odgovarajuće aktivnosti u manifest fajlu bi bilo potrebno dodati:

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category
android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

Android sistem testira svaki Intent objekat sa Intent filterom poređenjem tri elementa: akcije, podataka i kategorije. Da bi aktivnost prihvatila određeni Intent objekat, mora proći sva tri testa, inače Android sistem neće proslediti Intent objekat toj komponenti. Svaka komponenta može deklarisati i više filtera. Deo manifest fajla jedne socijalne aplikacije za deljenje različitog sadržaja bi mogao da izgleda ovako:

```
<activity android:name="MainActivity">
    <!—Ova aktivnost je inicijalna i ulaz u aplikaciju, treba
da se prikaze u app launcher-u -->
    <intent-filter>
        <action   android:name="android.intent.action.MAIN"
/>
        <category
android:name="android.intent.category.LAUNCHER" /
>
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!—Ova aktivnost prihvata akciju "SEND" sa tekstualnim
podacima -->
    <intent-filter>
```

```
  <category android:name="android.intent.category.DEFAULT"/>
  <data android:mimeType="text/plain"/>
  </intent-filter>
  <!—Ovaaktivnostprihvataakcije"SEND"
                                                              i
"SEND_MULTIPLE" sa multimedijom -->
<intent-filter>
<action android:name="android.intent.action.SEND"/>
<action android:name="android.intent.action.SEND_MULTIPLE"/>
<category android:name="android.intent.category.DEFAULT"/>
<data android:mimeType="application/vnd.google.panorama360+jpg"/>
<data android:mimeType="image/*"/>
<data android:mimeType="video/*"/>
</intent-filter>
</activity>
```

Prva aktivnost pod nazivom MainActivity je ulazna u aplikaciju, odnosno pokreće se kada korisnik pokrene aplikaciju dodirom na ikonicu. ACTION_MAIN označava da je u pitanju ulazna tačka u aplikaciju. CATEGORY_LAUNCHER označava da ikonica ove aktivnosti treba da se izlistava u listi aplikacija sistema. Ova dva atributa se moraju ovako upariti da bi se aktivnost prikazala u listi aplikacija. Druga aktivnost, ShareActivity, služi za deljenje teksta i multimedije. Korisnici mogu da dođu do ove aktivnosti ili navigacijom iz MainActivity, ili iz druge aplikacije prilikom implicitne namere koja prođe jedan od dva navedena Intent filtera.

Kada Android sistem dobije implicitnu nameru, on traži najadekvatniju aktivnost za dati Intent objekat poređenjem tog objekta sa Intent filterima u manifest fajlovima aplikacija. Ovaj proces se naziva rezolucija intenta, i podrazumeva poređenje po tri osnovna atributa: action, data (i URI i type) i category.

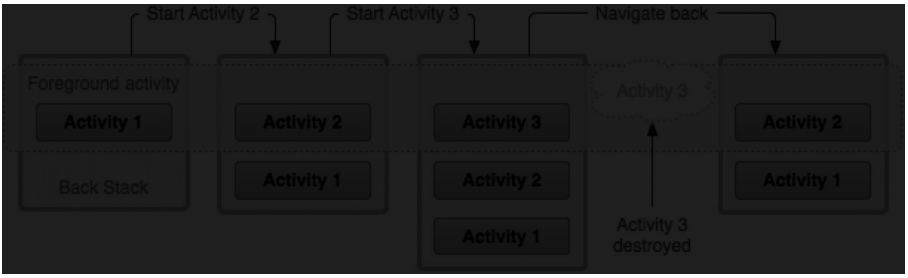
5.7. Back Stack

Android aplikacije se u opštem slučaju sastoje od više aktivnosti. Aktivnosti se projektuju na takav način da svaka treba da obavi neki specifičan vid interakcije sa korisnikom. U interakciju sa korisnikom spada i pokretanje nove aktivnosti iz trenutne aktivnosti, pri čemu nova aktivnost može biti deo aplikacije, ali i ne mora, kao što je objašnjeno kroz implicitne namere.

Sve aktivnosti sa kojima korisnik vrši interakciju dok obavlja određen posao (na primer slanje i provera mejlova, pregled neke od socijalnih mreža ili nešto treće) se zapravo posmatraju kao jedna celina. Ovakva kolekcija aktivnosti se u

Android sistemu naziva zadatak (engl. *task*). Sve ove aktivnosti su uređene u strukturi u

obliku steka, pod nazivom back stack, po redosledu pojedinačnog pokretanja svake aktivnosti. Stek je struktura koja je uređena po LIFO principu (engl. *Last In First Out*), odnosno poslednja dodata aktivnost će biti prva uklonjena iz steka. Kada korisnik otvori novu aplikaciju, kreira se novi task, pokreće se njena inicijalna aktivnost i stavlja na stek. Kada se iz te aktivnosti otvori nova aktivnost, ona se dodaje na vrh steka i dobija fokus. Prethodna aktivnost ostaje na steku u stopiranom stanju. Ukoliko korisnik pritisne Back dugme, trenutna aktivnost se uklanja sa vrha steka i uništava, a prethodna aktivnost dobija fokus i nastavlja sa interakcijom. Aktivnosti u steku se nikada ne preuređuju, već se samo mogu dodavati i uklanjati sa vrha steka. Na slici 5.12 prikazan je primer kretanja korisnika kroz aktivnosti i stanje steka u svakom trenutku.



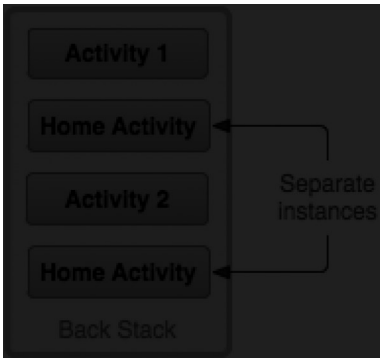
Slika 5.12, princip rada back stack-a
(izvor: https://developer.android.com/guide/components/activities/tasks-and-back-stack)

Kada korisnik pritisne Back dugme, aktivnost se uklanja sa vrha steka i uništava, pri čemu na vrh steka i u prvi plan izlazi prethodna aktivnost, sve dok se korisnik ne vrati na glavni ekran (ili bilo koju drugu aktivnost koja se izvršavala u trenutku pokretanja taska). Kada se sve aktivnosti uklone iz steka, task više ne postoji.

Jedan task se ponaša kao celina i pri prelasku u pozadinu, u slučajevima da korisnik odluči da pokrene novu aplikaciju koja izlazi u prvi plan, ili ukoliko ode na glavni ekran telefona pritiskom na Home dugme. Sve aktivnosti koje pripadaju tom tasku su u tom slučaju stopirane, ali back stack ostaje sačuvan, kako bi korisnik mogao da nastavi gde je stao kada odluči da vrati task u prvi plan (slika 5.13).

Slika 5.13, primer dva taska, pri čemu je task B u fokusu i vrši interakciju sa korisnikom, dok je task A u pozadini sa kompletno očuvanim back stack-om (izvor: https://developer.android.com/guide/components/activities/tasks-and-back-stack)

Posebna pažnja se mora obratiti na činjenicu da se aktivnosti u okviru back stack- a nikada ne preuređuju. Zbog toga se može desiti da u back stack-u postoji više instanci jedne iste aktivnosti u određenim okolnostima (slika 5.14). Primer je slučaj da aplikacija dozvoljava pokretanje određene aktivnosti iz više drugih aktivnosti, pri čemu se svaki put ta aktivnost kreira i stavlja na vrh back stack-a.



Slika 5.14, slučaj više instanci jedne aktivnosti u okviru back stack-a (izvor: https://developer.android.com/guide/components/activities/tasks-and-back-stack)

Opisano ponašanje back stack-a i taska je odgovarajući za većinu aplikacija. Ipak, moguće je napraviti odstupanje od definisanog načina rada, upotrebom atributa u manifestu aplikacije ili putem flag-ova u Intent objektu koji se prosleđuje aktivnosti. Bez dubljeg ulaženja u detalje, navodimo samo da su podržani sledeći scenariji:

- Nova aktivnost se pokreće u posebnom tasku umesto u već postojećem

- Prilikom pokretanja aktivnosti, podiže se već postojeća instanca umesto kreiranja nove na vrhu back stack-a.
- Kada korisnik napusti zadatak, sve aktivnosti se uklanjaju iz back stack-a, osim određene korene aktivnosti

5.8. Pitanja za vežbu

- Šta je aktivnost?
- Da li je potrebno deklarisati aktivnost u manifestu aplikacije?
- Koja su osnovna stanja u kojima aktivnost može da bude?
- Opisati ukratko životni ciklus aplikacije.
- Detaljno objasniti životni ciklus aktivnosti.
- Koje su callback metode životnog ciklusa?
- Između kojih callback metoda je aktivnost vidljiva?
- Koju metodu sistem poziva kada aktivnost počinje da gubi fokus?
- Koja metoda se poziva kada aktivnost više nije vidljiva?
- Šta se tipično radi u onCreate() metodi?
- Koji je redosled poziva callback metoda prilikom pokretanja aktivnosti?
- Kako se čuva stanje korisničkog interfejsa aktivnosti?
- Kako se rekreira sačuvano stanje korisničkog interfejsa aktivnosti?
- Šta je Bundle objekat i kako se koristi?
- Kako se stavljaju vrednosti u Bundle objekat?
- Kako se vade vrednosti iz Bundle objekta?
- Šta se dešava prilikom promene orijentacije ekrana?
- Kako se iz jedne aktivnosti pokreće druga aktivnost?
- Šta je namera (Intent)?
- Koji tipovi namera postoje?
- Objasniti detaljno razliku između eksplicitne i implicitne namere.
- Kako se mogu podaci iz jedne aktivnosti proslediti u drugu aktivnost?
- Šta se sve može pokrenuti sa Intent objektom?
- Koja metoda se koristi za pokretanje nove aktivnosti?
- Koja metoda se koristi za pokretanje nove aktivnosti ukoliko nova aktivnost treba da vrati rezultat prvoj?
- Kako se deklariše da aktivnost prima imlicitne namere?
- Šta je IntentFilter?
- Kako se vrši rezolucija Intenta?
- Objasniti princip rada Back Stack-a.

5.9. Zadaci za vežbu

Primer 1: Napraviti korisnički interfejs u kome korisnik unosi svoje ime, adresu i telefon. Raspored komponenti treba da bude linearni vertikalni. Na ekranu postoji i dugme na čiji klik aktivnost treba da preuzme sve unete podatke, i da ih prikaže u tekstualnom polju na ekranu. U aktivnosti dohvatiti reference na sve objekte korisničkog interfejsa, implementirati programski osluškivač za onclick događaj na dugmetu, preuzeti sve podatke sa korisničkog interfejsa i ispisati ih u pomenutom tekstualnom polju.

Rešenje:

Zadati korisnički interfejs se može implementirati na sledeći način, kako je definisano u activity_main.xml fajlu:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
android:layout_width="match_parent"
    android:layout_height="match_parent">
    <EditText
        android:id="@+id/inputIme"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Unesi ime"/>
    <EditText
        android:id="@+id/inputAdresa"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Unesi adresu"/>
    <EditText
        android:id="@+id/inputTelefon"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Unesi telefon"/>
    <Button
        android:id="@+id/buttonPosalji"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Posalji"/>
    <TextView
        android:id="@+id/labelUnetiPodaci"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20dp"
```

```
        android:lines="20" />
</LinearLayout>
```

Kod zadate aktivnosti, koja uzima reference na sve objekte i postavlja osluškivač u okviru posebno izdvojene initComponents() metode, a zatim nakon klika na dugme ispisuje sve podatke u tekstualnom polju, dat je sadržajem MainActivity.java klase:

```
package com.example.student.myapplication;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity
implements View.OnClickListener {

    private EditText inputIme;
    private EditText inputAdresa;
    private EditText inputTelefon;
    private Button buttonPosalji;
    private TextView labelUnetiPodaci;

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initComponents();
    }

    private void initComponents(){
        //dohvatanje referenci na objekte
        inputIme = findViewById(R.id.inputIme);
        inputAdresa = findViewById(R.id.inputAdresa);
        inputTelefon = findViewById(R.id.inputTelefon);
        labelUnetiPodaci =
findViewById(R.id.labelUnetiPodaci);
        buttonPosalji = findViewById(R.id.buttonPosalji);
        //postavljanje listenera
        buttonPosalji.setOnClickListener(this);
    }

    @Override
    public void onClick(View view) {
        //kliknuto dugme, pokupiti sve unete vrednosti iz
```

```
polja
        String unetoIme = inputIme.getText().toString();
        String unetaAdresa =
inputAdresa.getText().toString();
        String unetiTelefon =
inputTelefon.getText().toString();
        //postaviti u prikaz
        String podaci = "Uneti su podaci:\n" +
                unetoIme + "\n" +
                unetaAdresa + "\n" +
                unetiTelefon + "\n";
        labelUnetiPodaci.setText(podaci);
    }
}
```

Korisnički interfejs aplikacije nakon pokretanja prikazan je na slici 5.15 levo. Nakon unosa podataka i klika na dugme, uneti podaci se ispisuju u tekstualnom polju kao na slici 5.15 desno.

Slika 5.15, Prikaz aktivnosti koja preuzima podatke sa korisinčkog interfejsa i ispisuje ih u tekstualnom polju.

Primer 2: Napraviti aplikaciju koja dinamički kreira korisnički interfejs u obliku liste kontakata. Kontakti mogu da budu jednog od četiri moguća tipa (telefonski broj, Viber, Facebook ili Mail), a za svaki od tipova je u dinamički kreiranoj listi potrebno postaviti drugačiju ikonicu. Potrebno je kreirati posebni raspored komponenti koji će definisati izgled svakog pojedinačnog unosa.

Rešenje: Na početku, potrebno je definisati izgled korisničkog interfejsa. U ovom slučaju, odabran je ScrollView, unutar kojeg se nalazi linearni vertikalni raspored u koji će se programski ubaciti pojedinačni kontakti. Pojedinačni kontakti su takođe definisani sopstvenim rasporedom komponenti, koji se nalazi u posebnom xml fajlu. Sadržaj activity_main.xml fajla dat je u nastavku:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

tools:context="com.example.mzivkovic.myapplication.MainActi
vity">

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:id="@+id/mainScrollView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical" />
    </ScrollView>
 </RelativeLayout>
```

Svaki pojedinačni kontakt je definisan sopstvenim rasporedom komponenti. Ovaj raspored sadrži jedan ImageView, u kome se prikazuje ikonica odgovarajućeg tipa kontakta, i dva tekstualna polja u kojima se nalaze ime kontakta i odgovarajuća vrednost (broj telefona, mail adresa ili slično, u zavisnosti od tipa kontakta). Sadržaj je definisan u contact_view.xml fajlu na sledeći način:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="100sp">
```

```
    <ImageView
        android:id="@+id/imageIkonica"
        android:layout_width="100sp"
        android:layout_height="100sp"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_centerVertical="true"
        android:layout_margin="5sp"
        app:srcCompat="@drawable/user" />

    <TextView
        android:id="@+id/labelaIme"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_alignTop="@+id/imageIkonica"
        android:layout_toEndOf="@+id/imageIkonica"
        android:layout_toRightOf="@+id/imageIkonica"
        android:textSize="18sp" />

    <TextView
        android:id="@+id/labelaVrednost"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/imageIkonica"
        android:layout_below="@+id/labelaIme"
        android:layout_marginTop="5sp"
        android:layout_toEndOf="@+id/imageIkonica"
        android:layout_toRightOf="@+id/imageIkonica" />
</RelativeLayout>
```

Pojedinačni kontakt u kodu je predstavljen objektom klase Kontakt.java. U pitanju je obična klasa koja predstavlja model podatka koji se koristi u okviru aktivnosti, a čiji je kod dat u nastavku:

```
package com.example.mzivkovic.myapplication;

public class Kontakt {

    public enum TipKontakta {EMAIL, PHONE, VIBER, FACEBOOK}

    private String ime, vrednost;
    private TipKontakta tipKontakta;

    public Kontakt(String ime, String vrednost,
TipKontakta tipKontakta) {
```

```
        this.ime = ime;
        this.vrednost = vrednost;
        this.tipKontakta = tipKontakta;
    }

    public String getIme() {
        return ime;
    }

    public String getVrednost() {
        return vrednost;
    }

    public TipKontakta getTipKontakta() {
        return tipKontakta;
    }
}
```

U realnom scenariju, lista kontakata bi se dovlačila preko nekog oblika API-ja (dohvatanje preko Interneta, čitanje iz baze i slično). U ovom primeru, API je modelovan klasom KontaktAPI koja samo programski kreira listu kontakata i vraća je kao rezultat. Sadržaj ove klase dat je u nastavku:

```
package com.example.mzivkovic.myapplication;

import java.util.LinkedList;

public class KontaktAPI {

    public static LinkedList<Kontakt> getMyContacts () {
        LinkedList<Kontakt> lista = new LinkedList<>();
        lista.add(new Kontakt ("Pera Peric",
"+3811111111555", Kontakt.TipKontakta.PHONE));
        lista.add(new Kontakt ("Miodrag Zivkovic",
"Miodrag.Zivkovic.viber", Kontakt.TipKontakta.VIBER));
        lista.add(new Kontakt ("Jovan Jovanovic",
"jovan.jovanovic55@singidunum.ac.rs",
Kontakt.TipKontakta.EMAIL));
        lista.add(new Kontakt ("Mika Peric",
"+3811111311555", Kontakt.TipKontakta.FACEBOOK));
        lista.add(new Kontakt ("Miodrag Jovanovic",
"Miodrag.Jovanovic.viber", Kontakt.TipKontakta.VIBER));
        lista.add(new Kontakt ("Jovan Jovic",
"jovan.jovic32@singidunum.ac.rs",
Kontakt.TipKontakta.EMAIL));
        lista.add(new Kontakt ("Pera Petrovic",
"+3811111211555", Kontakt.TipKontakta.PHONE));
        lista.add(new Kontakt ("Miodrag Petrovic",
```

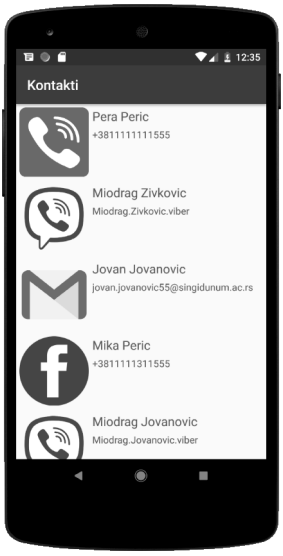
```
"mzz@singidunum.ac.rs", Kontakt.TipKontakta.EMAIL));
        lista.add(new Kontakt ("Jovan Jocic",
"jovan.jocic33@singudunum.ac.rs",
Kontakt.TipKontakta.EMAIL));
        return lista;
    }
}
```

Na kraju, aktivnost treba da dohvati generisanu listu kontakata, i da za svaki pojedinačni kontakt generiše njegov prikaz i smesti ga u ScrollView u activity_main.xml fajlu. Ovaj primer podrazumeva da se u direktorijumu res/drawable nalaze odgovarajuće ikonice za svaki tip kontakta. Kod klase MainActivity.java dat je u nastavku:

```
package com.example.mzivkovic.myapplication;

import android.content.Context;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;
import android.widget.TextView;

import java.util.LinkedList;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        LinkedList<Kontakt> lista =
KontaktAPI.getMyContacts();

        LinearLayout mainScrollView =
findViewById(R.id.mainScrollView);
        LayoutInflater inflater = (LayoutInflater)
getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        for (Kontakt kontakt : lista){
            RelativeLayout item = (RelativeLayout)
inflater.inflate(R.layout.contact_view, null);
            ((TextView)
item.findViewById(R.id.labelaIme)).setText(kontakt.getIme()
);
```

```
            ((TextView)
item.findViewById(R.id.labelaVrednost)).setText(kontakt.
getVrednost());

            ImageView imageIkonica = (ImageView)
item.findViewById(R.id.imageIkonica);
            switch (kontakt.getTipKontakta()){
                case EMAIL:
imageIkonica.setImageResource(R.drawable.email);
                    break;
                case PHONE:
imageIkonica.setImageResource(R.drawable.phone);
                    break;
                case VIBER:
imageIkonica.setImageResource(R.drawable.viber);
                    break;
                case FACEBOOK:
imageIkonica.setImageResource(R.drawable.facebook);
            }
            mainScrollView.addView(item);
        }
    }
}
```

Nakon pokretanja aplikacije, u okviru onCreate() metode se dinamički kreira raspored komponenti za svakog pojedinačnog kontakta i smešta u korisnički interfejs. Izgled aplikacije je dat na slici 5.16.



Slika 5.16, dinamički kreiran korisnički interfejs

Primer 3: Potrebno je napraviti aplikaciju za studente, koja od korisnika u prvom ekranu traži da unese odgovarajuće podatke o sebi (korisničko ime, šifru, odabir fakulteta koji studira, status zaposlenja i datum rođenja). Nakon klika na dugme, svi ovi podaci se uzimaju iz korisničkog interfejsa, pakuju u Intent objekat i šalju drugoj aktivnosti koja će ih prihvatiti i prikazati na ekranu.

Rešenje: Ovaj primer ilustruje eksplicitnu nameru, pošto se svi podaci uneti u prvoj aktivnosti prosleđuju drugoj aktivnosti, koju glavna aktivnost pokreće nakon klika na dugme. Korisnički interfejs prve aktivnosti dat je u fajlu activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
tools:context="com.example.mzivkovic.myapplication.MainActi
vity">
    <ScrollView
        android:id="@+id/scrollView2"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical" >

            <TextView
                android:id="@+id/labelUsername"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Username" />

            <EditText
                android:id="@+id/inputUsername"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
```

```
            android:ems="10"
            android:inputType="textPersonName" />

        <TextView
            android:id="@+id/labelPassword"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20sp"
            android:text="Password" />

        <EditText
            android:id="@+id/inputPassword"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:inputType="textPassword" />

        <TextView
            android:id="@+id/labelFaculty"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20sp"
            android:text="Faculty" />

        <Spinner
            android:id="@+id/inputFaculty"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

        <TextView
            android:id="@+id/labelEmployed"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20sp"
            android:text="Employed" />

        <Switch
            android:id="@+id/inputEmployed"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Are you employed?" />

        <TextView
            android:id="@+id/labelBirthdate"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20sp"
            android:text="Birth date" />
```

```
            <DatePicker
                android:id="@+id/inputBirthdate"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:datePickerMode="spinner"
                android:calendarViewShown="false" />

            <Button
                android:id="@+id/buttonSend"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="20sp"
                android:text="Send" />
        </LinearLayout>
    </ScrollView>
</android.support.constraint.ConstraintLayout>
```

Glavna aktivnost nakon klika na dugme sakuplja sve unete vrednosti od strane korisnika, pakuje ih u Intent objekat i nakon toga startuje drugu aktivnost. U pitanju je eksplicitna namera, pošto se druga aktivnost poziva eksplicitno po imenu aktivnosti u Intent objektu koji se prosleđuje kao argument startActivity() metode. Kod MainActivity.java klase dat je u nastavku:

```
package com.example.mzivkovic.myapplication;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.Switch;
import android.widget.TextView;
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity
implements View.OnClickListener {

    private EditText inputUsername;
    private EditText inputPassword;
    private Spinner inputFaculty;
    private Switch inputEmployed;
    private DatePicker inputBirthday;
    private Button buttonSend;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initComponents();
    }

    public void initComponents(){
        //inicijalizacija liste fakulteta za spinner
        inputFaculty = findViewById(R.id.inputFaculty);
        ArrayList<String> facultyList = new ArrayList<>();
        facultyList.add("Fakultet za informatiku i
racunarstvo");
        facultyList.add("Tehnicki fakultet");
        facultyList.add("Poslovni fakultet");

        ArrayAdapter<String> adapter = new
ArrayAdapter<String>(this,
android.R.layout.simple_spinner_dropdown_item,
facultyList);

        inputFaculty.setAdapter(adapter);

        inputUsername = findViewById(R.id.inputUsername);
        inputPassword = findViewById(R.id.inputPassword);
        inputEmployed = findViewById(R.id.inputEmployed);
        inputBirthday = findViewById(R.id.inputBirthdate);
        buttonSend = findViewById(R.id.buttonSend);
        buttonSend.setOnClickListener(this);
    }

    @Override
    public void onClick(View view) {
        if (view.getId()== R.id.buttonSend){
            Intent intent = new Intent(this,
ConfirmationActivity.class);

            String username =
inputUsername.getText().toString();
            String password =
inputPassword.getText().toString();
            String faculty = (String)
inputFaculty.getSelectedItem();
            boolean employed = inputEmployed.isChecked();

            String birthdayString = String.format("%4d-%2d-
```

```
%2d", inputBirthday.getYear(), inputBirthday.getMonth() +
1, inputBirthday.getDayOfMonth());
            Bundle extras = new Bundle();
            extras.putString("username", username);
            extras.putString("password", password);
            extras.putString("faculty", faculty);
            extras.putBoolean("employed", employed);
            extras.putString("birthday", birthdayString);

            intent.putExtras(extras);
            startActivity(intent);
        }

    }
}
```

Druga aktivnost takođe ima svoj fajl sa rasporedom komponenti. Ovaj fajl je u ovom primeru jednostavan, sadrži samo tekstualno polje gde se prikazuju svi podaci koji su primljeni kroz Intent objekat. Raspored komponenti druge aktivnosti dat je sadržajem activity_confirmation.xml fajla u nastavku:

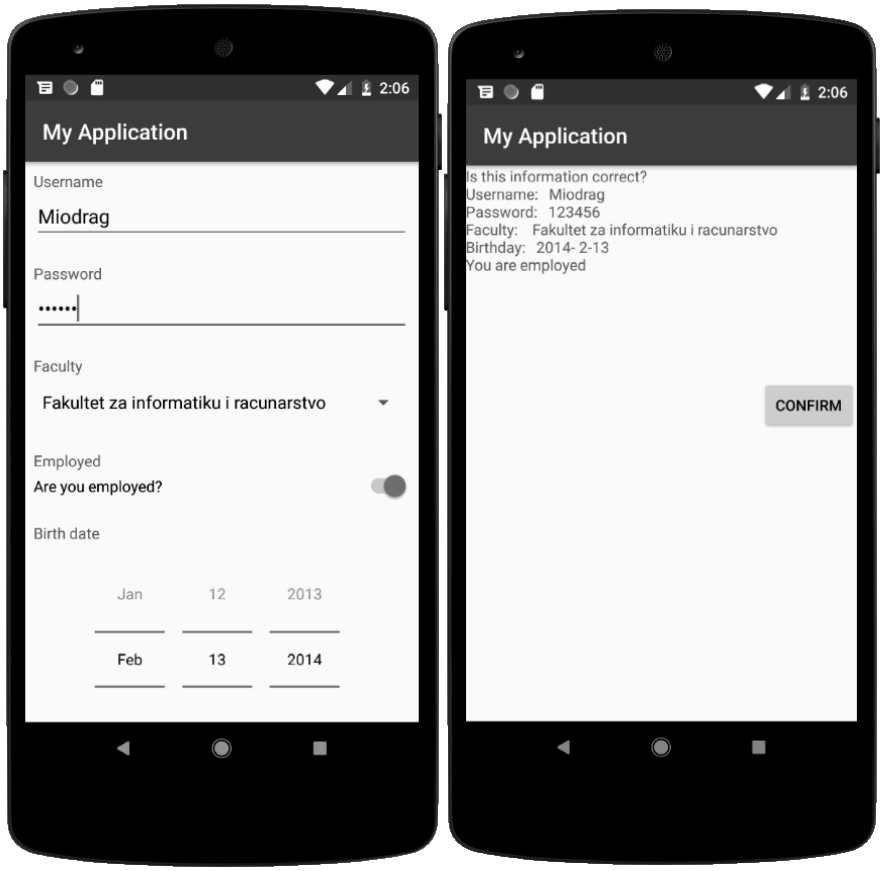
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

tools:context="com.example.mzivkovic.myapplication.Confirma
tionActivity">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/labelMessage"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:lines="10"
            android:text="Message" />

        <Button
            android:id="@+id/buttonConfirm"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentEnd="true"
```

```
            android:layout_alignParentRight="true"
            android:layout_below="@+id/labelMessage"
            android:layout_marginTop="30sp"
            android:text="Confirm" />
    </RelativeLayout>
</android.support.constraint.ConstraintLayout>
```

Kod same aktivnosti dat je u klasi ConfirmationActivity.xml:

```
package com.example.mzivkovic.myapplication;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class ConfirmationActivity extends AppCompatActivity
{

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_confirmation);

        Bundle extras = getIntent().getExtras();
        String username = extras.getString("username");
        String password = extras.getString("password");
        String faculty = extras.getString("faculty");
        Boolean employed = extras.getBoolean("employed");
        String birthday = extras.getString("birthday");
        String message = "Is this information correct? \n";
        message += "Username:    " + username + "\n";
        message += "Password:    " + password + "\n";
        message += "Faculty:     " + faculty + "\n";
        message += "Birthday:    " + birthday + "\n";
        message += (employed)? "You are employed \n" : "You
are not employed \n";
  ((TextView)
findViewById(R.id.labelMessage)).setText(message);

    }
}
```

Prikaz rada aplikacije dat je na slici 5.17. Sa leve strane slike prikazana je prva aktivnost, koja od korisnika traži da unese sve zahtevane informacije, a klikom na dugme pokreće drugu aktivnost i šalje joj sve unete podatke. Sa desne strane iste slike prikazana je druga aktivnost, koja ispisuje sve podatke koje primila kroz Intent objekat od prve aktivnosti.

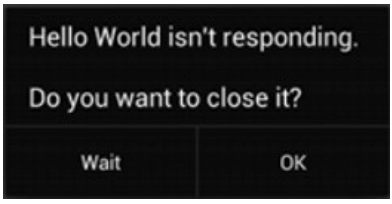


Slika 5.17, primer aplikacije koja koristi eksplicitnu nameru za pokretanje druge aktivnosti u okviru iste aplikacije

6. Niti

Svaka Android aplikacija se, nakon pokretanja, pokreće u okviru novog Linux procesa sa jednom niti u kojoj se ta aplikacija izvršava. Ova nit se naziva glavna nit (engl. *main thread*), i u njoj se izvršavaju sve komponente aplikacije. Poznata je još pod nazivom korisnička nit (engl. *UI thread* - nit korisničkog interfejsa), zbog toga što je zadužena za obradu događaja koje korisnik generiše prilikom interakcije sa korisničkim interfejsom aplikacije, kao i za iscrtavanje događaja. Drugim rečima, ova nit je veoma bitna zato što se u okviru nje odvija kompletna interakcija aplikacije sa korisnikom, odnosno sa komponentama Android UI toolkit (komponente definisane u android.widget i android.view paketima).

Android sistem je dizajniran na takav način da se sve komponente istog procesa instanciraju i izvršavaju u jednoj – korisničkoj niti. Sistem neće samostalno pokrenuti novu odvojenu nit za novu instancu neke komponente. Dodatno, sve callback metode (poput onClick(), onKeyUp() itd.) se okidaju i izvršavaju u ovoj niti. Ako bi aplikacija u okviru ovih callback metoda izvršavala neki dug i složen posao, to bi drastično uticalo na performanse same aplikacije. Uopšteno gledano, bilo kakva dugotrajna operacija koja se izvršava u korisničkoj niti, poput dohvatanja podataka preko mreže ili pristupa bazi podataka, će blokirati kompletno korisnički interfejs. Kada je korisnička nit blokirana, to znači da se ne generišu niti obrađuju događaji, kao i da se korisnički interfejs ne iscrtava – drugim rečima ekran aplikacije je zamrznut, a korisnik ne može da izvrši nikakvu interakciju sa njim. Situaciju dodatno pogoršava sam Android sistem – ukoliko se primeti da je korisnička nit neke aplikacije blokirana više od pet sekundi, korisniku će biti prikazan „Application not responding“ dijalog (slika 6.1). Ovaj zloglasni dijalog je poznat i pod skraćenim imenom ANR. Nije potrebno dodatno naglašavati koliko ANR može imati loše posledice po zadovoljstvo korisnika, koji može da odluči i da obriše aplikaciju. Dakle, programer mora uvek biti svestan činjenice da će Android sistem prikazati ANR u slučaju da aplikacija ne može da odgovori na korisnički unos.



Slika 6.1, Application not responding dijalog

Još jedno ograničenje je postavljeno od strane samog Android UI toolkit-a, koji nije bezbedno koristiti u višenitnom sistemu (nije *thread-safe*). Posledica po

programera se ogleda u tome da se elementima korisničkog interfejsa ne sme

pristupati iz neke druge niti, već isključivo iz korisničke niti. Navedeni problemi se mogu sumirati u dva veoma jednostavna pravila koje programer mora poštovati u svakom trenutku:

- Korisnička nit se ne sme blokirati,
- Komponentama interfejsa se ne sme pristupati van korisničke niti.

Kako se izbegava ANR? Svaka dugotrajna operacija koju aplikacija obavlja u korisničkoj niti može izazvati ANR dijalog. Zbog toga, sve metode koje se izvršavaju u okviru korisničke niti moraju da budu što brže i da obavljaju što je moguće manje posla u samoj korisničkoj niti. Naravno, to ne znači da aplikacija uopšte ne sme da obavlja dugotrajne operacije, koje su vrlo često neophodne i predstavljaju samu srž i osnovnu funkcionalnost date aplikacije. Takve operacije treba izvršiti u odvojenoj niti, kako se ne bi ugrozila brzina odziva korisničkog interfejsa aplikacije. Iz drugih niti nije dozvoljeno ažurirati korisnički interfejs – to se sme obavljati samo iz korisničke niti. Android nudi nekoliko opcija kako se iz drugih niti može pristupiti korisničkoj niti, kroz sledeće metode:

- Activity.runOnUiThread(Runnable) – izvršava zadatu akciju u korisničkoj niti. Ukoliko je trenutna nit korisnička nit, akcija se izvršava odmah. Ukoliko trenutna nit nije korisnička nit, akcija se stavlja u red za čekanje na izvršavanje u korisničkoj niti.
- View.post(Runnable) – dodaje akciju u message queue, pri čemu će se Runnable akcija izvršiti u korisničkoj niti.
- View.postDelayed(Runnable, long) - dodaje akciju u message queue, pri čemu će se Runnable akcija izvršiti u korisničkoj niti nakon isteka zadatog vremena.

Na primer, implementacija data u nastavku je bezbedna u okruženju više niti – dugotrajna operacija obrade slike se vrši u odvojenoj niti, dok se ažuriranje ImageView komponente obavezno vrši iz korisničke niti.

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            // obrada koja moze dugo trajati
            final Bitmap bitmap =
                    obradiSliku("slika1.png");
            imageView.post(new Runnable() {
                public void run() {
                    imageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

Ovo je klasičan primer tzv. radne niti (engl. *worker thread*), čiji je posao da obavi dugotrajnu operaciju odvojeno od niti korisničkog interfejsa. Iako nema ništa pogrešno u ovoj implementaciji, to ipak nije preporučeni način rada sa nitima, bar u okviru Android sistema. Ovakav kod teži da se usložnjava ukoliko se poveća kompleksnost operacije koju treba da izvrši, a samim tim postaje težak za održavanje. Kako bi se malo olakšao razvoj, može se koristiti Handler objekat unutar radne niti za isporuku rezultata operacije korisničkoj niti. Najbolje rešenje, međutim, je upotreba klase AsyncTask, koja drastično olakšava izvršavanje radne niti koja u nekom trenutku mora da komunicira sa korisničkim interfejsom. Ovaj pristup je preporučen od strane Android zajednice i opisan je u nastavku ovog poglavlja.

6.1. AsyncTask

Klasa AsyncTask dozvoljava izvršavanje asinhronih zadataka na korisničkom interfejsu aplikacije. Ona objedinjuje dva procesa time što izvršava blokirajuću operaciju u radnoj niti, a objavljuje rezultate na korisničkoj niti, bez potrebe da programer samostalno rukuje nitima. To je preporučeni način korišćenja niti za obavljanje dužih pozadinskih operacija koje je kasnije potrebno objaviti na ekranu. Sama klasa je osmišljena kao pomoćna klasa bazirana na Thread i Handler klasama. Treba napomenuti da za dugotrajne pozadinske operacije (koji traju duže od desetak sekundi), Android zajednica i dalje preporučuje tradicionalan pristup i upotrebu java.util.concurrent paketa.

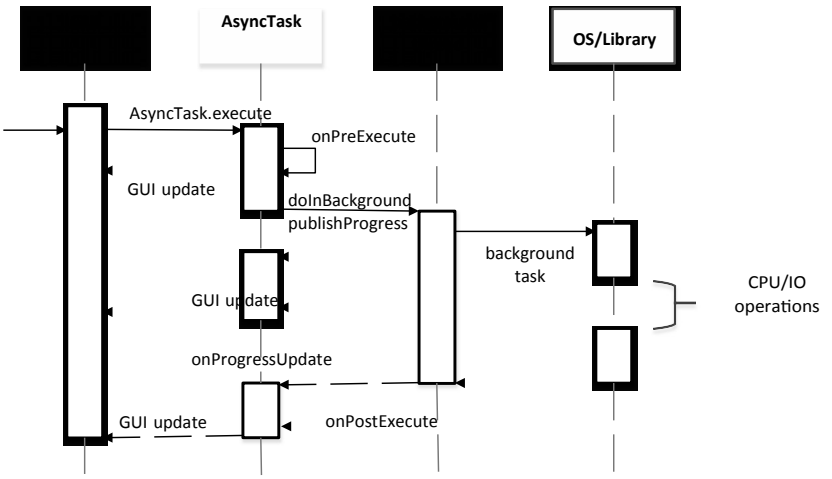
Praktično gledano, AsyncTask se koristi tako što se kreira nova klasa, izvedena iz AsyncTask klase, a zatim implementira nekoliko callback metoda. Sam AsyncTask je definisan sa tri generička tipa i četiri koraka (metode koje je potrebno implementirati). Generički tipovi se definišu u obliku AsyncTask<Params, Progress, Result>, pri čemu je njihova uloga sledeća:

- Params – označava tip ulaznih parametara koje AsyncTask prihvata pre izvršavanja.
- Progress – označava tip jedinice koja služi da se objavi progres za vreme izvršavanja samog taska.
- Result – označava tip rezultata operacije koja se izvršava u pozadini.

Nije uvek potrebno koristiti sve tipove, ali se to mora označiti sa tipom Void. Na primer, AsyncTask<String, Void, Bitmap> označava da task uzima String kao ulazni parametar, ne prikazuje progres i vraća Bitmap kao rezultat rada nakon završetka operacije. Osim potrebnih parametara, AsyncTask je određen i sa četiri koraka kroz koje prolazi dok se izvršava. Svaki od koraka je povezan i sa metodom koju je potrebno implementirati. Koraci su, redom:

- onPreExecute() – metoda koja se poziva u okviru korisničke niti, pre izvršavanja samog taska. U ovom koraku potrebno je pripremiti sve što je tasku potrebno, poput inicijalizacije početnog progresa na vrednost 0.
- doInBackground(Params…) – metoda se izvršava u pozadinskoj niti, a poziva se neposredno nakon izvršavanja metode onPreExecute(). Prosleđuju joj se parametri zadati u definiciji AsyncTask<Params, Progress, Result>. Ova metoda izvršava dugotrajnu operaciju, čiji se rezultat mora vratiti u okviru ovog koraka, i koji će biti prosleđen poslednjem koraku, odnosno metodi onPostExecute(Result). Metoda doInBackground(Params…) može da koristi metodu publishProgress(Progress…) za objavljivanje progresa izvršavanja operacije.
- onProgressUpdate(Progress…) – ova metoda se okida u okviru korisničke niti nakon poziva metode publishProgress(Progress…). Tačno vreme izvršavanja ove metode nije definisano. Pošto se izvršava u korisničkoj niti, ona sme da pristupa korisničkom interfejsu i da prikaže u nekom obliku progres pozadinske operacije, poput progresne linije, procenta ili slično.
- onPostExecute(Result) – metoda se poziva u okviru korisničke niti nakon završetka pozadinske operacije. Rezultat operacije joj se prosleđuje kao parametar.

Task se pokreće pozivom metode execute(Params…) iz korisničke niti. Tok jednog asinhronog taska prikazan je na slici 6.2. Jednom pokrenut task se po potrebi može otkazati pozivom cancel(boolean) metode. U tom slučaju će, umesto onPostExecute() metode, biti pozvana metoda onCancelled(). Status otkazivanja se može po potrebi proveriti pozivom metode isCancelled().



Slika 6.2, dijagram toka AsyncTask-a

Kako bi se AsyncTask ispravno koristio, potrebno je pridržavati se nekoliko pravila:

- Instanca AsyncTask se mora kreirati u okviru korisničke niti.
- execute(Params…) se mora pozvati u okviru korisničke niti.
- Nije potrebno ručno pozivati metode onPreExecute, doInBackground, onProgressUpdate i onPostExecute.
- Task se može izvršiti samo jednom, svaki naknadni pokušaj izvršavanja istog taska će rezultirati izbacivanjem izuzetka.

Tipičan primer AsyncTaska jeste preuzimanje fajlova preko mreže. Ovakva operacija može da traje nepredviđeno dugo, zbog lošeg kvaliteta mreže. U slučaju da se ovakva operacija radi u okviru korisničke niti, za sve vreme dok traje preuzimanje fajlova korisnik ne bi mogao da vrši interakciju sa aplikacijom. Tipična implementacija preuzimanja fajlova data je u nastavku. Definisana je klasa ZadatakPreuzimanjeFajlova, koja proširuje AsyncTask. Parametri AsyncTaska dati su u obliku AsyncTask<URL, Integer, Long>. URL objekti su potrebni za preuzimanje fajlova, Integer parametar služi za objavljivanje progresa operacije, a Long predstavlja tip povratnog rezultata. Ovaj primer implementacije podrazumeva da postoji implementirana klasa Downloader koji u sebi sadrži statičku metodu downloadFile(Url), čijim pozivom se preuzima fajl na lokaciji zadatoj pomoću vrednosti Url parametra. Takođe, pretpostavlja se da ova metoda kao povratnu vrednost vraća broj preuzetih bajtova.

```
private class ZadatakPreuzimanjeFajlova extends
AsyncTask<URL, Integer, Long> {
     protected Long doInBackground(URL... urls) {
         int brojFajlova = urls.length;
         long ukupnaVelicina = 0;
         for (int i = 0; i < brojFajlova; i++) {
             ukupnaVelicina +=
Downloader.downloadFile(urls[i]);
             publishProgress((int) ((i / (float)
brojFajlova) * 100));
             // prekinuti operaciju ukoliko je pozvana
metoda cancel()
             if (isCancelled()) break;
         }
         return ukupnaVelicina;
     }
     protected void onProgressUpdate(Integer... progress) {
         setProgressPercent(progress[0]);
     }
     protected void onPostExecute(Long rezultat) {
         showDialog("Preuzeto " + rezultat + " bajtova");
     }
 }
```

6.2. Pitanja za vežbu

- U koliko niti se u okviru svog procesa podrazumevano izvršava Android aplikacija?
- Šta je korisnička nit (main thread)? Šta se sve radi u okviru ove niti?
- Šta se dešava ukoliko se korisnička nit blokira? Detaljno objasniti slučaj da je nit na kratko blokirana (manje od 5 sekundi), i duže blokirana (duže od 5 sekundi).
- Šta je ANR dijalog?
- Da li se komponentama korisničkog interfejsa sme pristupati van korisničke niti?
- Koja su dva osnovna pravila po pitanju upotrebe korisničke niti, kojih se programeri obavezno moraju pridržavati?
- Navesti primere tipičnih operacija koje se izvršavaju u odvojenim nitima.
- Koje načine Android nudi drugim nitima za pristup korisničkoj niti?
- Objasniti koncept radne niti (worker thread).
- Koji je preporučeni način korišćenja niti u Androidu?
- Šta je AsyncTask i čemu služi?
- Objasniti detaljno svaki od gereničkih tipova definisanih zaglavljem AsyncTask<Params, Progress, Result>.
- Koji su osnovi koraci kroz koje AsyncTask prolazi? Koje metode im odgovaraju?
- Objasniti metodu onPreExecute().
- Objasniti metodu doInBackground().
- Objasniti metodu onPostExecute().
- Kako se objavljuje progres izvršavanja asinhronog zadatka?
- Da li je moguće prekinuti već pokrenuti AsyncTask?
- Da li je moguće reciklirati AsyncTask, odnosno, da li je moguće ponovo pokrenuti jednom izvršeni AsyncTask?
- U okviru koje niti se kreira instanca AsyncTask?
- Da li se metode onPreExecute, doInBackground, onProgressUpdate i onPostExecute mozivaju ručno ili ne?
- Navesti neke tipične primere kada je pogodno upotrebiti AsyncTask.

6.3. Zadaci za vežbu

Primer 1: Napraviti aplikaciju koja preuzima sliku sa Interneta upotrebom AsyncTask. Preuzimanje slike se inicira klikom na odgovarajuće dugme, a nakon preuzimanja slika se pokazuje na korisničkom interfejsu.

Rešenje: Korisnički interfejs se sastoji od jednog dugmeta kojim se inicira preuzimanje slike, i jednog ImageView objekta u koji će biti smeštena slika. Sadržaj activity_main.xml fajla je dat u nastavku:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <Button
        android:id="@+id/buttonDownload"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Preuzmi sliku" />
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:contentDescription="Ovde ce biti
        prikazana
slika" />
</LinearLayout>
```

Pošto aplikacija treba da preuzme sliku sa Interneta, potrebna joj je odgovarajuća dozvola u manifestu aplikacije (android.permission.Internet). Sadržaj AndroidManifest.xml fajla dat je sa:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.student.asynctask">
    <uses-permission
android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
```

```
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

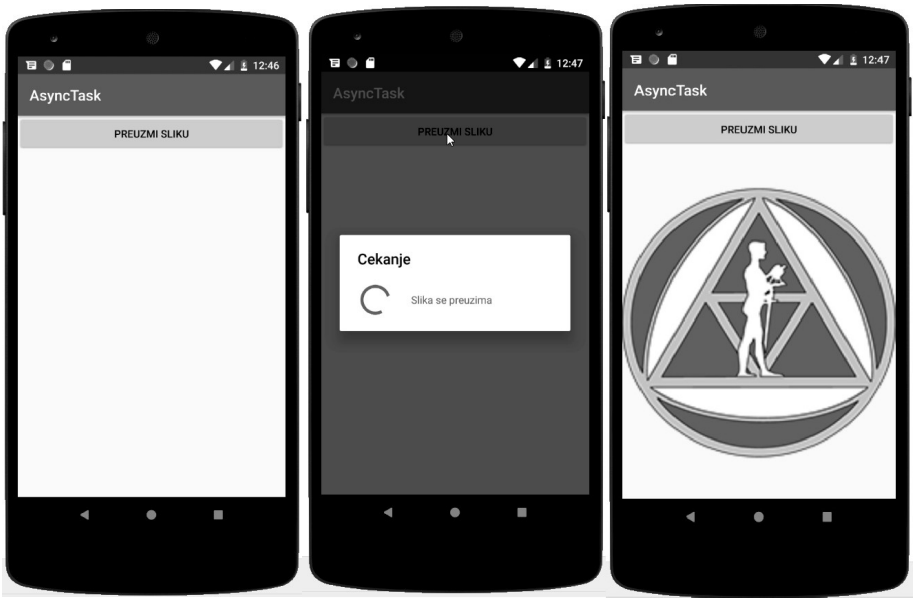
Aktivnost preuzima sliku preko AsyncTask-a, pošto se može desiti da ova akcija potraje. Ako bi se preuzimanje vršilo u okviru korisničke niti, moglo bi doći do blokiranja korisničkog interfejsa i ANR dijaloga. Kod MainActivity.java dat je sa:

```
package com.example.student.asynctask;

import android.app.ProgressDialog;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

import java.io.InputStream;
import java.net.HttpURLConnection;

public class MainActivity extends AppCompatActivity
implements View.OnClickListener {
    private ImageView preuzetaSlika;
    private ProgressDialog dijalogZaCekanje;
    private Button buttonPreuzmiSliku;
    private static String url =
"https://repository.singidunum.ac.rs/images/2014/06/thumb/s
ingidunum-logo.png";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initComponents();
    }

    private void initComponents(){
```

```
        buttonPreuzmiSliku =
findViewById(R.id.buttonDownload);
        preuzetaSlika = findViewById(R.id.imageView);
        buttonPreuzmiSliku.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        new TaskPreuzimanjeSlike().execute(url);
    }

    private class TaskPreuzimanjeSlike extends
AsyncTask<String,Void,Bitmap> {
        @Override
        protected Bitmap doInBackground(String... param) {
            try{
                java.net.URL url = new
java.net.URL(param[0]);
                HttpURLConnection connection =
(HttpURLConnection) url.openConnection();
                connection.setDoInput(true);
                connection.connect();
                InputStream input =
connection.getInputStream();
                Bitmap myBitmap =
BitmapFactory.decodeStream(input);
                input.close();
                return myBitmap;
            } catch (Exception e) {
                // rukovanje izuzetkom
            }
            return null;
        }
        @Override
        protected void onPreExecute() {
            dijalogZaCekanje =
ProgressDialog.show(MainActivity.this,
                    "Cekanje", "Slika se preuzima");
        }

        @Override
        protected void onPostExecute(Bitmap result) {
            preuzetaSlika.setImageBitmap(result);
            dijalogZaCekanje.dismiss();
        }
    }
}
```

Asinhroni task je definisan u obliku AsyncTask<String,Void,Bitmap>. Prvi parametar je String, kroz koji se prosleđuje url adresa na kojoj se slika za preuzimanje nalazi. Drugi parametar je Void, pošto se progres ne objavljuje, već se samo vrti dijalog za čekanje. Treći parametar je tipa Bitmap, pošto se slika nakon preuzimanja dekoduje u Bitmap i tako vraća kao rezultat.

Nakon pokretanja aplikacije, korisniku se prikazuje interfejs kao na slici 6.3 (levo). Klikom na dugme za preuzimanje slike se inicira asinhrono preuzimanje pomoću prethodno definisanog AsyncTask-a. Dok se slika preuzima korisniku se prikazuje dijalog za čekanje, koji je prikazan na slici 6.3 u sredini. Nakon preuzimanja slike, ona se smešta u ImageView i prikazuje korisniku (6.3 desno). Lokacija odakle se slika preuzima zadata je statičkom String promenljivom url. Ukoliko korisnik želi da preuzme neku drugu sliku, dovoljno je staviti njen URL u ovu promenljivu.



Slika 6.3, Preuzimanje slike pomoću AsyncTask

Primer 2:

Implementirati Android aplikaciju koja će preko AsyncTask-a kontaktirati API na adresi http://199.188.100.46/singidunum/android/faculties.json. GET zahtev na ovu adresu vraća listu fakulteta u obliku JSON fajla, koji je prikazan sledećim listingom:

```
[{
```

```
}, {
        "name": "Fakultet za informatiku i računarstvo",
        "acronym": "FIR",
        "website": "http://fir.singidunum.ac.rs"
}, {
        "name": "Poslovni fakultet u Beogradu",
        "acronym": "PFB",
        "website": "http://pfb.singidunum.ac.rs"

        "name": "Fakultet za turistički i hotelijerski
menadžment",
        "acronym": "FTHM",
        "website": "http://fthm.singidunum.ac.rs"
}, {
        "name": "Tehnički fakultet",
        "acronym": "TF",
        "website": "http://tf.singidunum.ac.rs"
}]
```

Kada se dohvati lista fakulteta, potrebno je parsirati JSON u domenske objekte (klasa Faculty), i nakon toga prikazati korisniku na ekranu.

Rešenje: Korisnički interfejs aplikacije treba da sadrži jedno dugme, čijim klikom se asinhrono učitava lista fakulteta sa API-ja. Osim dugmeta, potrebno je i jedno tekstualno polje za prikaz dovučenih rezultata. Kod activity_main.xml fajla dat je sa:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/buttonFaculties"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Dohvati listu fakulteta"/>
    <TextView
        android:id="@+id/labelFaculties"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="15dp"
        android:lines="20" />
</LinearLayout>
```

Pošto je za kontaktiranje API-ja neophodno da aplikacija ima pristup internet, mora se zahtevati odgovarajuća dozvola u okviru AndroidManifest.xml fajla:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.student.myapplication">

    <uses-permission
android:name="android.permission.INTERNET"></uses-
permission>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Dohvatanje liste fakulteta se odvija u odvojenoj niti u okviru AsyncTask. Za komunikaciju između više niti može se koristiti objekat klase Handler. Putem Handler objekta se iz jedne niti može zakazati akcija koju je potrebno izvršiti u drugoj niti. Handler se zasniva na mehanizmu prosleđivanja poruka. U ovom primeru se koristi naša klasa ReadDataHandler, koja je izvedena iz Handler klase. Proširena klasa dodatno sadrži polje kroz koje će se glavnoj korisničkoj niti proslediti odgovor dobijen od servera. Sadržaj ReadDataHandler.java fajla je dat u nastavku:

```
package com.example.student.myapplication;

import android.os.Handler;

public class ReadDataHandler extends Handler {
    private String json;

    public String getJson() {
        return json;
    }
```

```
    public void setJson(String json) {
        this.json = json;
    }
}
```

Dalje, potrebno je implementirati klasu koja će predstavljati domenske objekte. Podaci iz JSON-a nisu previše laki za manipulaciju ukoliko bi se čuvali samo kao Stringovi. Potrebna je klasa čiji će objekat sadržati sva polja koja se nalaze u primljenom JSON-u, pošto se objektima u Java kodu mnogo lakše manipuliše. Klasa Faculty.java predstavlja domenske objekte, a dodatno pruža i uslužne metode za parsiranje jednog fakulteta, kao i parsiranje niza fakulteta. Kod ove klase dat je u nastavku:

```
package com.example.student.myapplication;

import org.json.JSONArray;
import org.json.JSONObject;
import java.util.LinkedList;

public class Faculty {

    private String name, acronym, website;

    public Faculty(){

    }

    public Faculty(String name, String acronym, String
website) {
        this.name = name;
        this.acronym = acronym;
        this.website = website;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAcronym() {
        return acronym;
    }

    public void setAcronym(String acronym) {
        this.acronym = acronym;
```

```
    }

    public String getWebsite() {
        return website;
    }

    public void setWebsite(String website) {
        this.website = website;
    }

    //parsiranje jednog objekta
    public static Faculty parseJSONObject (JSONObject
object){
        Faculty faculty = new Faculty();
        try {
            if (object.has("name")) {
                faculty.setName(object.getString("name"));
            }
            if (object.has("acronym")){

faculty.setAcronym(object.getString("acronym"));
            }
            if (object.has("website")){
           faculty.setWebsite(object.getString("website"));
            }

        } catch (Exception e){

        }
        return faculty;
    }

    //parsiranje niza objekata, koje se zapravo zasniva
na iterativnom parsiranju pojedinacnih objekata
    public static LinkedList<Faculty> parseJSONArray
(JSONArray array){
        LinkedList<Faculty> list = new LinkedList<>();
        try {
            for (int i = 0; i < array.length(); i++) {
                Faculty faculty =
parseJSONObject(array.getJSONObject(i));
                list.add(faculty);
            }
        } catch (Exception e){ }
        return list;
    }
}
```

Za komunikaciju sa API-jem kreira se klasa Api, koja će kroz poziv metode getJSON() kreirati asinhroni poziv ka serveru kako bi se dohvatila lista fakulteta. Dohvaćeni podaci se smeštaju u prosleđeni parametar ReadDataHandler, koji će moći da bude pročitan u korisničkoj niti. Kod klase Api.java dat je u nastavku:

```
package com.example.student.myapplication;

import android.os.AsyncTask;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class Api {

    public static void getJSON (String url, final
ReadDataHandler rdh){
        AsyncTask<String,Void,String> task = new
AsyncTask<String, Void, String>() {
            @Override
            protected String doInBackground(String...
strings) {
                String response = "";

                //treba nam http konekcija
                try {
                    URL link = new URL(strings[0]);
                    HttpURLConnection con =
(HttpURLConnection) link.openConnection();

                    BufferedReader br = new
BufferedReader(new
InputStreamReader(con.getInputStream()));
                    //cita se red po red iz br
                    String red;
                    while ((red = br.readLine())!=null){
                        response += red + "\n";
                    }

                    br.close();
                    con.disconnect(); // ne mora se pisati
- Android sam ubija konekciju

                } catch (Exception e){
                    response = "[]";
                }
                return response;
```

```
            }

            @Override
            protected void onPostExecute(String response) {
                rdh.setJson(response);
                rdh.sendEmptyMessage(0);
            }
        };

        task.execute(url);
    }

}
```

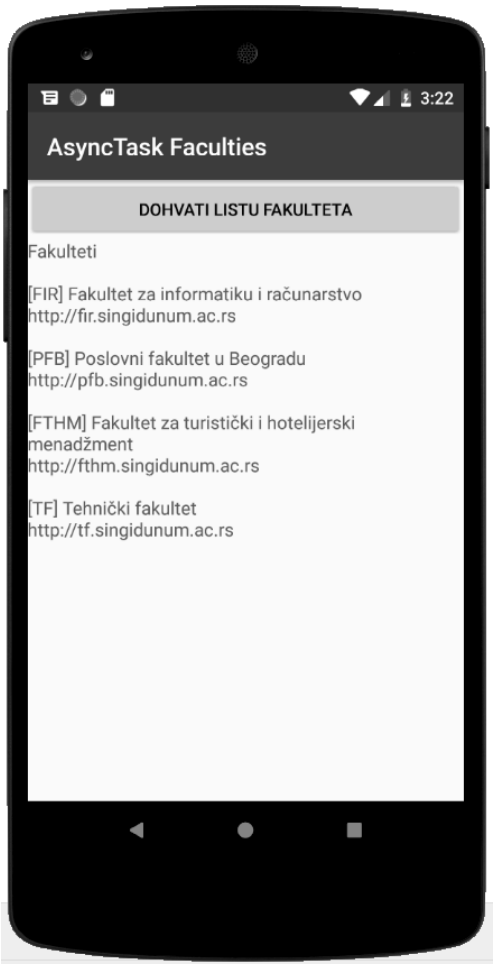
Na kraju, u glavnoj aktivnosti se klikom na dugme okida poziv statičke metode Api klase getJSON(), kojoj se prosleđuje objekat ReadDataHandler, kroz koji će stići odgovor kada podaci budu asinhrono skinuti sa servera, pri čemu se koristi mehanizam poruka. Kod MainActivity.java dat je u nastavku:

```
package com.example.student.myapplication;

import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import org.json.JSONArray;

import java.util.LinkedList;

public class MainActivity extends AppCompatActivity
implements View.OnClickListener{

    private Button buttonFaculties;
    private TextView labelFaculties;

    @Override
    protected void onCreate(Bundle savedInstanceState)
        { super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initComponents();

    }

    private void initComponents(){
```

```
        buttonFaculties =
findViewById(R.id.buttonFaculties);
        labelFaculties = findViewById(R.id.labelFaculties);
        buttonFaculties.setOnClickListener(this);
    }

    @Override
    public void onClick(View view) {

Api.getJSON("http://199.188.100.46/singidunum/android/facul
ties.json", new ReadDataHandler(){
            @Override
            public void handleMessage(Message msg) {
                String odgovor = getJson();
                try {
                    JSONArray array = new
JSONArray(odgovor);
                    LinkedList<Faculty> faculties =
Faculty.parseJSONArray(array);


labelFaculties.setText("Fakulteti\n\n");
                    for (Faculty f : faculties) {
                        String prikaz = "[" +
f.getAcronym() + "] " + f.getName() + "\n" + f.getWebsite()
+ "\n\n";
                        labelFaculties.append(prikaz);
                    }

                } catch (Exception e) {

                }
            }
        });

        labelFaculties.setText("Ucitava se...");

    }

}
```

Metoda handleMessage() objekta tipa ReadDataHandler se poziva kao odgovor na poziv metode sendEmptyMessage() u okviru onPostExecute() metode AsyncTask- a, čime se signalizira da je odgovor servera spreman i da se nalazi u polju json ReadDataHandler objekta. Dalje, u kodu handleMessage() metode se parsira dobijeni odgovor pozivom uslužnih metoda iz Faculty klase, a parsirani

odgovor smešta u objekte tipa Faculty. Na kraju se parsirana lista objekata prikazuje na korisničkom interfejsu, kao što je prikazano na slici 6.4.



Slika 6.4, Prikaz dohvaćene liste fakulteta preko AsyncTask-a